Cyclone V Avalon-ST Interface for PCIe Solutions

User Guide

Last updated for Altera Complete Design Suite: 14.1
UG-01110_avst
2014.12.15

Subscribe
Send Feedback

101 Innovation Drive
San Jose, CA 95134
www.altera.com

Datasheet 1

2014.12.15

UG-01110_avst Subscribe Send Feedback

Cyclone V Avalon-ST Interface for PCIe Datasheet

Altera® Cyclone® V FPGAs include a configurable, hardened protocol stack for PCI Express® that is compliant with *PCI Express Base Specification 2.1 or 3.0*. The Hard IP for PCI Express using the Avalon Streaming (Avalon-ST) interface is the most flexible variant. However, this variant requires a thorough understanding of the PCIe® Protocol. The following figure shows the high-level modules and connecting interfaces for this variant.

Figure 1-1: Cyclone V PCIe Variant with Avalon-ST Interface

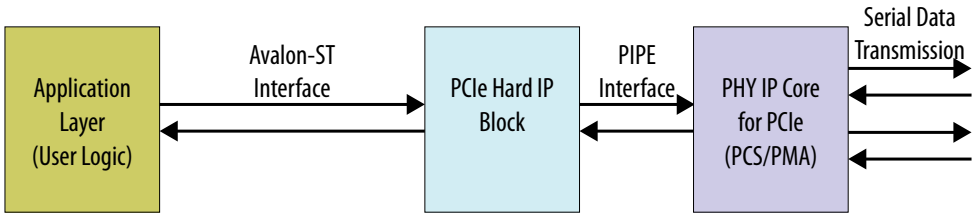


Table 1-1: PCI Express Data Throughput

The following table provides bandwidths for a single transmit (TX) or receive (RX) channel. The numbers double for duplex operation. Gen1 and Gen2 use 8B/10B encoding which introduces a 20% overhead.

	Link Width		
	×1	×2	×4
PCI Express Gen1 (2.5 Gbps)	2	4	8
PCI Express Gen2 (5.0 Gbps)	4	8	16

Refer to the *PCI Express High Performance Reference Design* for more information about calculating bandwidth for the hard IP implementation of PCI Express in many Altera FPGAs.

Related Information

- **PCI Express Base Specification 2.1 or 3.0**

© 2014 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

- PCI Express High Performance Reference Design
- Creating a System with Qsys

Features

New features in the Quartus® II 14.1 software release:

- Reduced Quartus II compilation warnings by 50%.

The Cyclone V Hard IP for PCI Express supports the following features:

- Complete protocol stack including the Transaction, Data Link, and Physical Layers implemented as hard IP.
- Support for ×1, ×2, and ×4 configurations with Gen1 and Gen2 lane rates for Root Ports and Endpoints.
- Dedicated 16 KByte receive buffer.
- Optional hard reset controller for Gen2.
- Optional support for Configuration via Protocol (CvP) using the PCIe link allowing the I/O and core bitstreams to be stored separately.
- Qsys example designs demonstrating parameterization, design modules, and connectivity.
- Extended credit allocation settings to better optimize the RX buffer space based on application type.
- Multi-function support for up to eight Endpoint functions.
- Optional end-to-end cyclic redundancy code (ECRC) generation and checking and advanced error reporting (AER) for high reliability applications.

  Easy to use:

  - Flexible configuration.
  - Substantial on-chip resource savings and guaranteed timing closure.
  - No license requirement.
  - Example designs to get started.

Table 1-2: Feature Comparison for all Hard IP for PCI Express IP Cores

The table compares the features of the four Hard IP for PCI Express IP Cores.

Feature	Avalon-ST Interface	Avalon-MM Interface	Avalon-MM DMA
IP Core License	Free	Free	Free
Native Endpoint	Supported	Supported	Supported
Legacy Endpoint (1)	Supported	Not Supported	Not Supported
Root port	Supported	Supported	Not Supported
Gen1	×1, ×2, ×4	×1, ×2, ×4	Not Supported

(1) Not recommended for new designs.

Feature	Avalon-ST Interface	Avalon-MM Interface	Avalon-MM DMA
Gen2	×1, ×2, ×4	×1, ×2, ×4	×4
64-bit Application Layer interface	Supported	Supported	Not supported
128-bit Application Layer interface	Supported	Supported	Supported
Transaction Layer Packet type (TLP)	• Memory Read Request • Memory Read Request-Locked • Memory Write Request • I/O Read Request • I/O Write Request • Configuration Read Request (Root Port) • Configuration Write Request (Root Port) • Message Request • Message Request with Data Payload • Completion Message • Completion with Data • Completion for Locked Read without Data	• Memory Read Request • Memory Write Request • I/O Read Request—Root Port only • I/O Write Request—Root Port only • Configuration Read Request (Root Port) • Configuration Write Request (Root Port) • Completion Message • Completion with Data • Memory Read Request (single dword) • Memory Write Request (single dword)	• Memory Read Request • Memory Write Request • Completion Message • Completion with Data
Payload size	128–512 bytes	128 or 256 bytes	128 or 256 bytes
Number of tags supported for non-posted requests	32 or 64	16	16
62.5 MHz clock	Supported	Supported	Not Supported
Multi-function	Supports up to 8 functions	Supports single function only	Supports single function only
Out-of-order completions (transparent to the Application Layer)	Not supported	Supported	Supported

Feature	Avalon-ST Interface	Avalon-MM Interface	Avalon-MM DMA
Requests that cross 4 KByte address boundary (transparent to the Application Layer)	Not supported	Supported	Supported
Polarity Inversion of PIPE interface signals	Supported	Supported	Supported
ECRC forwarding on RX and TX	Supported	Not supported	Not supported
Number of MSI requests	1, 2, 4, 8, or 16	1, 2, 4, 8, or 16	1, 2, 4, 8, or 16
MSI-X	Supported	Supported	Supported
Legacy interrupts	Supported	Supported	Supported
Expansion ROM	Supported	Not supported	Not supported

The purpose of the *Cyclone V Avalon-ST Interface for PCI e Solutions User Guide* is to explain how to use this and not to explain the PCI Express protocol. Although there is inevitable overlap between these two purposes, this document should be used in conjunction with an understanding of the *PCI Express Base Specification*.

Note: This release provides separate user guides for the different variants. The *Related Information* provides links to all versions.

Related Information

- **V-Series Avalon-MM DMA Interface for PCIe Solutions User Guide**
- **Cyclone V Avalon-MM Interface for PCIe Solutions User Guide**
- **Cyclone V Avalon-ST Interface for PCIe Solutions User Guide**

Release Information

Table 1-3: Hard IP for PCI Express Release Information

Item	Description
Version	14.1
Release Date	December 2014

Item	Description
Ordering Codes	No ordering code is required
Product IDs	There are no encrypted files for the Cyclone V Hard IP for PCI Express. The Product ID and Vendor ID are not required because this IP core does not require a license.
Vendor ID	

Device Family Support

Table 1-4: Device Family Support

Device Family	Support
Cyclone V	Final. The IP core is verified with final timing models. The IP core meets all functional and timing requirements for the device family and can be used in production designs.
Other device families	Refer to the *Related Information* below for other device families:

Related Information

- **Arria V Avalon-MM Interface for PCIe Solutions User Guide**
- **Arria V Avalon-ST Interface for PCIe Solutions User Guide**
- **Arria V GZ Avalon-MM Interface for PCIe Solutions User Guide**
- **Arria V GZ Avalon-ST Interface for PCIe Solutions User Guide**
- **Arria 10 Avalon-MM Interface for PCIe Solutions User Guide**
- **Arria 10 Avalon-MM DMA Interface for PCIe Solutions User Guide**
- **Arria 10 Avalon-ST Interface for PCIe Solutions User Guide**
- **IP Compiler for PCI Express User Guide**
- **Stratix V Avalon-MM Interface for PCIe Solutions User Guide**
- **Stratix V Avalon-ST Interface for PCIe Solutions User Guide**
- **Stratix V Avalon-ST Interface with SR-IOV for PCIe Solutions User Guide**

Configurations

The Cyclone V Hard IP for PCI Express includes a full hard IP implementation of the PCI Express stack comprising the following layers:

- Physical (PHY), including:
  - Physical Media Attachment (PMA)
  - Physical Coding Sublayer (PCS)
- Media Access Control (MAC)
- Data Link Layer (DL)
- Transaction Layer (TL)

The Hard IP supports all memory, I/O, configuration, and message transactions. It is optimized for Altera devices. The Application Layer interface is also optimized to achieve maximum effective throughput. You can customize the Hard IP to meet your design requirements.

Figure 1-2: PCI Express Application with a Single Root Port and Endpoint

The following figure shows a PCI Express link between two Cyclone V FPGAs. One is configured as a Root Port and the other as an Endpoint.

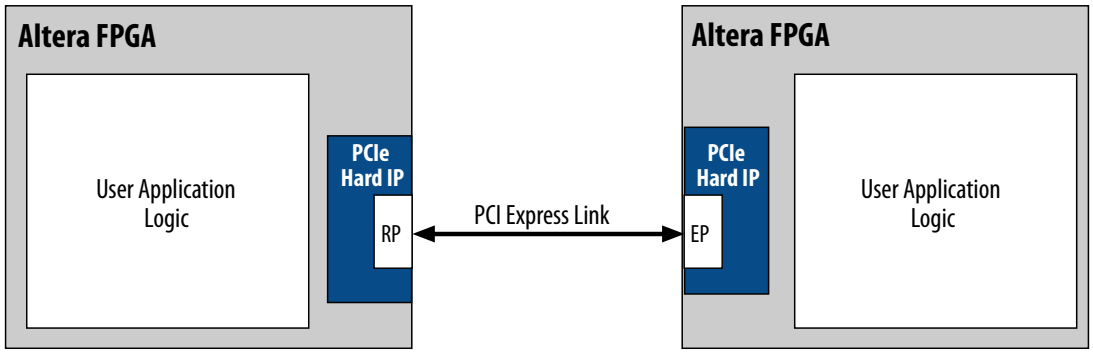


Figure 1-3: PCI Express Application with an Endpoint Using the Multi-Function Capability

The following figure shows a PCI Express link between two Altera FPGAs. One is configured as a Root Port and the other as a multi-function Endpoint. The FPGA serves as a custom I/O hub for the host CPU. In the Cyclone V FPGA, each peripheral is treated as a function with its own set of Configuration Space registers. Eight multiplexed functions operate using a single PCI Express link.

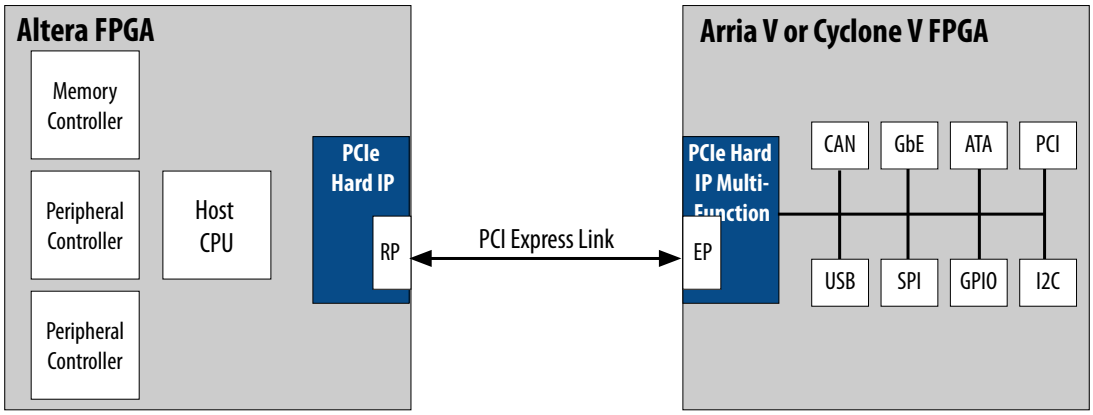
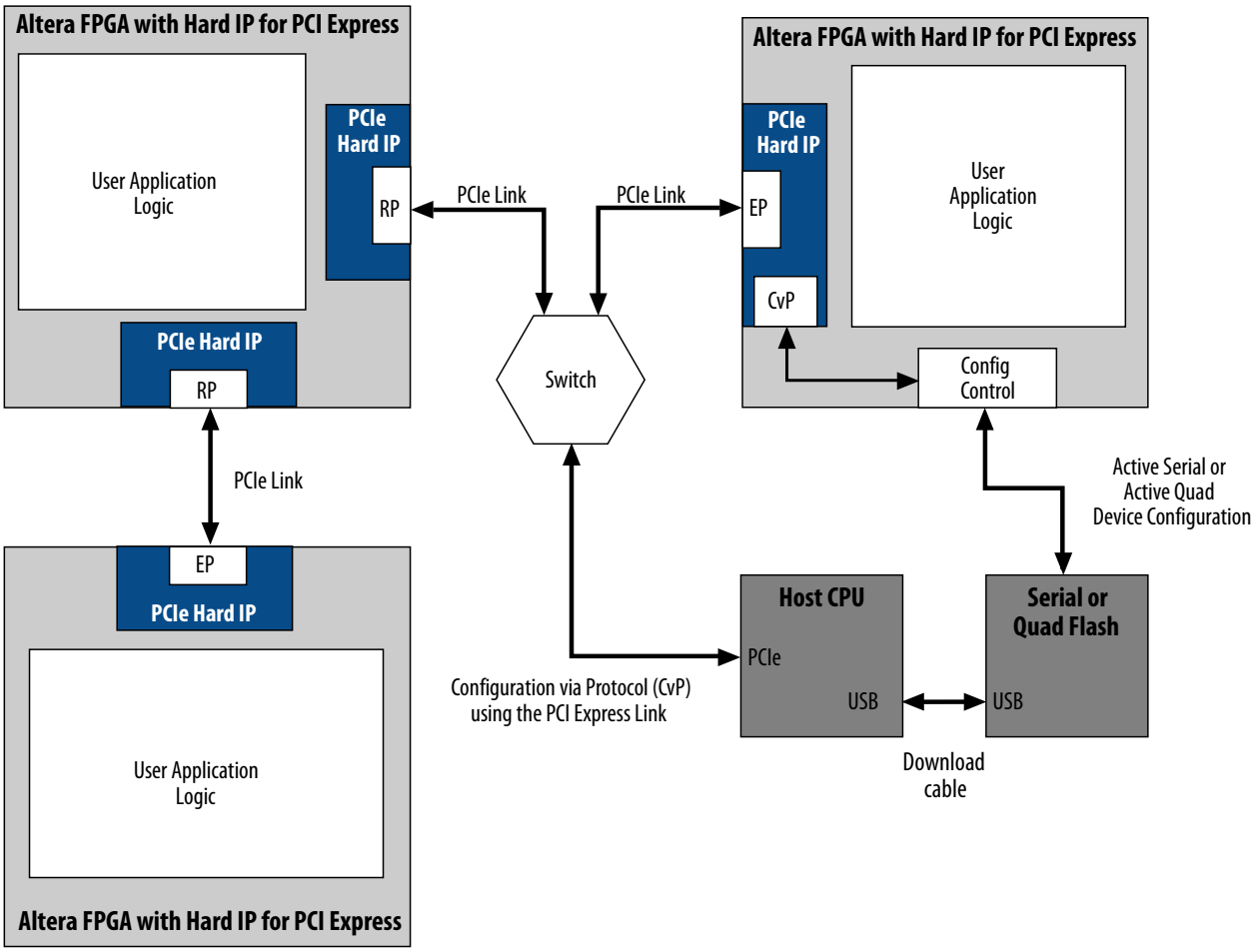


Figure 1-4: PCI Express Application Using Configuration via Protocol

The Cyclone V design below includes the following components:

- A Root Port that connects directly to a second FPGA that includes an Endpoint.
- Two Endpoints that connect to a PCIe switch.
- A host CPU that implements CvP using the PCI Express link connects through the switch. For more information about configuration over a PCI Express link below.



Related Information

Configuration via Protocol (CvP)Implementation in Altera FPGAs User Guide

Example Designs

Altera provides example designs to familiarize you with the available functionality. Each design connects the device under test (DUT) to an application (APPS) as the figure below illustrates. Certain critical parameters of the APPs component are set to match the values of DUT. If you change these parameters, you must change the APPs component to match. You can change the values for all other parameters of the DUT without editing the APPs component.

- Targeted Device Family
- Lanes
- Lane Rate
- Application Clock Rate
- Port type
- Application Interface

- Tags supported
- Maximum payload size
- Number of functions

The following example designs are available for the Cyclone V Hard IP for PCI Express. You can download them from the **<*install_dir*>/ ip/altera/altera_pcie/altera_pcie_hip_ast_ec/example_design/<*dev*>** directory:

- **pcie_de_gen1_x1_ast64.qsys**
- **pcie_de_gen1_x4_ast64.qsys**
- **pcie_de_rp_gen1_x4_ast64.qsys**

Click on the link below to get started with the example design provided in this user guide.

Related Information

Getting Started with the Cyclone V Hard IP for PCI Express on page 2-1

Debug Features

Debug features allow observation and control of the Hard IP for faster debugging of system-level problems.

Related Information

Debugging on page 17-1

IP Core Verification

To ensure compliance with the PCI Express specification, Altera performs extensive verification. The simulation environment uses multiple testbenches that consist of industry-standard bus functional models (BFMs) driving the PCI Express link interface. Altera performs the following tests in the simulation environment:

- Directed and pseudorandom stimuli are applied to test the Application Layer interface, Configuration Space, and all types and sizes of TLPs
- Error injection tests that inject errors in the link, TLPs, and Data Link Layer Packets (DLLPs), and check for the proper responses
- PCI-SIG® Compliance Checklist tests that specifically test the items in the checklist
- Random tests that test a wide range of traffic patterns

Altera provides the following two example designs that you can leverage to test your PCBs and complete compliance base board testing (CBB testing) at PCI-SIG.

Related Information

- **PCI SIG Gen3 x8 Merged Design - Stratix V**
- **PCI SIG Gen2 x8 Merged Design - Stratix V**

Compatibility Testing Environment

Altera has performed significant hardware testing to ensure a reliable solution. In addition, Altera internally tests every release with motherboards and PCI Express switches from a variety of manufacturers. All PCI-SIG compliance tests are run with each IP core release.

Performance and Resource Utilization

Because the PCIe protocol stack is implemented in hardened logic, it uses less than 1% of device resources.

Note: Soft calibration of the transceiver module requires additional logic. The amount of logic required depends on the configuration.

Related Information

Fitter Resources Reports

Recommended Speed Grades

Table 1-5: Cyclone V Recommended Speed Grades for Link Widths and Application Layer Clock Frequencies

Altera recommends setting the Quartus II Analysis & Synthesis Settings **Optimization Technique** to **Speed** when the Application Layer clock frequency is 250 MHz. For information about optimizing synthesis, refer to *Setting Up and Running Analysis and Synthesis* in Quartus II Help. For more information about how to effect the **Optimization Technique** settings, refer to *Area and Timing Optimization* in volume 2 of the *Quartus II Handbook*. .

Cyclone V Gen2 variants must use GT parts.

Link Rate	Link Width	Interface Width	Application Clock Frequency (MHz)	Recommended Speed Grades
Gen1	×1	64 bits	62.5[(2)],125	–6, –7,–8
	×2	64 bits	125	–6, –7,–8
	×4	64 bits	125	–6, –7,–8
Gen2	×1	64 bits	125	–7
	×2	64 bits	125	–7
	×4	128 bits	125	–7

(2) This is a power-saving mode of operation

Related Information

- **Area and Timing Optimization**
- **Altera Software Installation and Licensing Manual**
- **Setting up and Running Analysis and Synthesis**

Steps in Creating a Design for PCI Express

Before you begin

Select the PCIe variant that best meets your design requirements.

- Is your design an Endpoint or Root Port?
- What Generation do you intend to implement?
- What link width do you intend to implement?
- What bandwidth does your application require?
- Does your design require CvP?

1. Select parameters for that variant.
2. Simulate using an Altera-provided example design. All of Altera's PCI Express example designs are available under **<*install_dir*>/ip/altera/altera_pcie/**. Alternatively, create a simulation model and use your own custom or third-party BFM. The Qsys Generate menu generates simulation models. Altera supports ModelSim®-Altera for all IP. The PCIe cores support the Aldec RivieraPro, Cadence NCsim, Mentor Graphics ModelSim, and Synopsys VCS and VCS-MX simulators.
3. Compile your design using the Quartus II software. If the versions of your design and the Quartus II software you are running do not match, regenerate your PCIe design.
4. Download your design to an Altera development board or your own PCB. Click on the *All Development Kits* link below for a list of Altera's development boards.
5. Test the hardware. You can use Altera's SignalTap® II Logic Analyzer or a third-party protocol analyzer to observe behavior.
6. Substitute your Application Layer logic for the Application Layer logic in Altera's testbench. Then repeat Steps 3–6. In Altera's testbenches, the PCIe core is typically called the DUT (device under test). The Application Layer logic is typically called APPS.

Related Information

- **Parameter Settings** on page 3-1
- **Getting Started with the Cyclone V Hard IP for PCI Express**
- **All Development Kits**

Getting Started with the Cyclone V Hard IP for PCI Express

2

2014.12.15

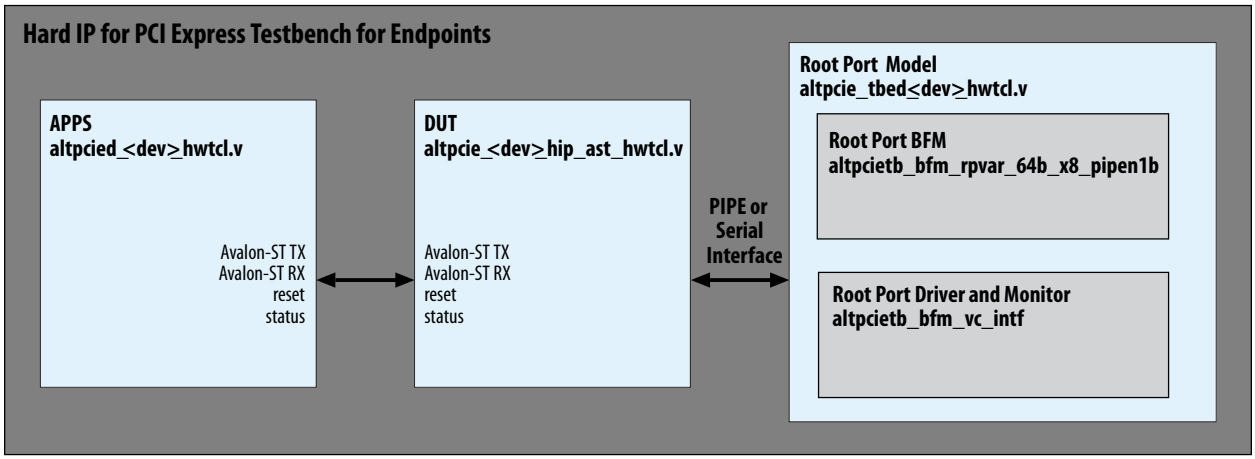
UG-01110_avst

Subscribe Send Feedback

This section provides instructions to help you quickly customize, simulate, and compile the Cyclone V Hard IP for PCI Express IP Core. When you install the Quartus II software you also install the IP Library. This installation includes design examples for Hard IP for PCI Express under the ***<install_dir>*/ip/altera/altera_pcie/** directory.

After you install the Quartus II software for 14.0, you can copy the design examples from the ***<install_dir>*/ip/altera/altera_pcie/altera_pcie_hip_ast_ed/example_designs/*<dev>*** directory. This walkthrough uses the Gen1 ×4 Endpoint, **pcie_de_gen1_x4_ast64.qsys**. The following figure illustrates the top-level modules of the testbench in which the DUT, a Gen1 Endpoint, connects to a chaining DMA engine, labeled APPS in the following figure, and a Root Port model. The simulation can use the parallel PHY Interface for PCI Express (PIPE) or serial interface.

Figure 2-1: Testbench for an Endpoint



Altera provides example designs to help you get started with the Cyclone V Hard IP for PCI Express IP Core. You can use example designs as a starting point for your own design. The example designs include scripts to compile and simulate the Cyclone V Hard IP for PCI Express IP Core. This example design provides a simple method to perform basic testing of the Application Layer logic that interfaces to the Hard IP for PCI Express.

© 2014 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

For a detailed explanation of this example design, refer to the *Testbench and Design Example* chapter. If you choose the parameters specified in this chapter, you can run all of the tests included in *Testbench and Design Example* chapter.

For more information about Qsys, refer to *System Design with Qsys* in the *Quartus II Handbook*. For more information about the Qsys GUI, refer to *About Qsys* in Quartus II Help.

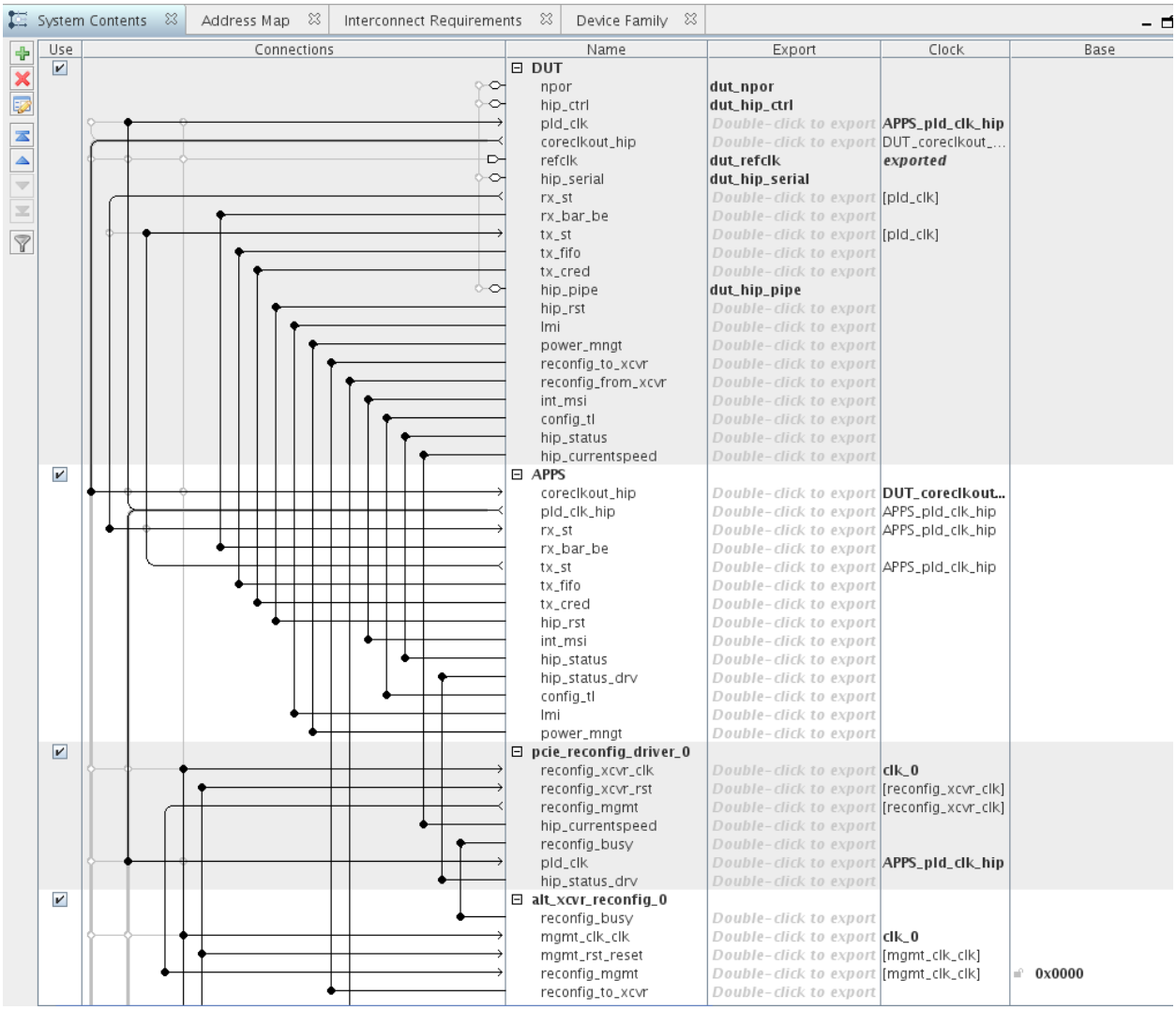
Related Information

- **System Design with Qsys**
- **About Qsys**

Qsys Design Flow

Copy the **pcie_de_gen1_x4_ast64.qsys** design example from the **<*install_dir*>/ip/altera/altera_pcie/altera_pcie/altera_pcie_hip_ast_ed/example_designs/<*dev*>** to your working directory. The following figure illustrates this Qsys system.

Figure 2-2: Complete Gen1 ×4 Endpoint (DUT) Connected to Example Design (APPS)



The example design includes the following components:

- DUT—This is Gen1 ×4 Endpoint. For your own design, you can select the data rate, number of lanes, and either Endpoint or Root Port mode.
- APPS—This DMA driver configures the DUT and drives read and write TLPs to test DUT functionality.
- pcie_reconfig_driver_0—This Avalon-MM master drives the Transceiver Reconfiguration Controller. The pcie_reconfig_driver_0 is implemented in clear text that you can modify if your design requires different reconfiguration functions. After you generate your Qsys system, the Verilog HDL for this component is available as: **<*working_dir*>/<*variant_name*>/testbench/<*variant_name*>_tb/simulation/submodules/altpcie_reconfig_driver.sv**.
- Transceiver Reconfiguration Controller—The Transceiver Reconfiguration Controller dynamically reconfigures analog settings to improve signal quality. For Gen1 and Gen2 data rates, the Transceiver Reconfiguration Controller must perform offset cancellation and PLL calibration.

Generating the Testbench

Follow these steps to generate the chaining DMA testbench:

1. On the **Generate** menu, select **Generate Testbench System**. Specify the parameters listed in the following table.

Table 2-1: Parameters to Specify on the Generation Tab in Qsys

Parameter	Value
Create testbench Qsys system	**Standard, BFMs for standard Qsys interfaces**
Create testbench simulation model	**Verilog**
Allow mixed-language simulation	Turn this option off
Output Directory	
Path	**<*working_dir*>/pcie_de_gen1_x4_ast64**
Testbench	**<*working_dir*>/pcie_de_gen1_x4_ast64/testbench**

2. Click the **Generate** button at the bottom of the Generation tab to create the testbench.

Simulating the Example Design

1. Start your simulation tool. This example uses the ModelSim® software.
2. From the ModelSim transcript window, in the testbench directory type the following commands:
   a. `do msim_setup.tcl`
   b. `ld_debug` (This command compiles all design files and elaborates the top-level design without any optimization.)
   c. `run -all`

The simulation includes the following stages:

- Link training
- Configuration
- DMA reads and writes
- Root Port to Endpoint memory reads and writes

Disabling Scrambling to Interpret TLPs at the PIPE Interface

1. Go to **<*project_directory*/<*variant*>/testbench/<*variant*>_tb/simulation/submodules/**.
2. Open **altpcietb_bfm_top_rp.v.**
3. Locate the declaration of `test_in[2:1]`. Set `test_in[2] = 1` and `test_in[1] = 0`. Changing `test_in[2] = 1` disables data scrambling on the PIPE interface.
4. Save **altpcietb_bfm_top_rp.v**.

Generating Quartus II Synthesis Files

1. On the **Generate** menu, select **Generate HDL**.
2. For **Create HDL design files for synthesis**, select **Verilog**.
   You can leave the default settings for all other items.
3. Click **Generate** to generate files for Quartus II synthesis.
4. Click **Finish** when the generation completes.

Understanding the Files Generated

Table 2-2: Overview of Qsys Generation Output Files

Directory	Description
***<testbench_dir>/<variant_name>*/synthesis**	Includes the top-level HDL file for the Hard IP for PCI Express and the **.qip** file that lists all of the necessary assignments and information required to process the IP core in the Quartus II compiler. Generally, a single **.qip** file is generated for each IP core.
***<testbench_dir>/<variant_name>*/synthesis/submodules**	Includes the HDL files necessary for Quartus II synthesis.
***<testbench_dir>/<variant_name>*/testbench**	Includes testbench subdirectories for the Aldec, Cadence, Synopsys, and Mentor simulation tools with the required libraries and simulation scripts.
<testbench_dir>/<variant_name>*/testbench*<cad_vendor>	Includes the HDL source files and scripts for the simulation testbench.

For a more detailed listing of the directories and files the Quartus II software generates, refer to *Files Generated for Altera IP Cores* in *Compiling the Design in the Qsys Design Flow*.

Understanding Physical Placement of the PCIe IP Core

For more information about physical placement of the PCIe blocks, refer to the links below. Contact your Altera sales representative for detailed information about channel and PLL usage.

Compiling the Design in the Quartus II Software

To compile the Qsys design example in the Quartus II software, you must create a Quartus II project and add your Qsys files to that project.

Complete the following steps to create your Quartus II project:

1. Click the **New Project Wizard** icon.
2. Click **Next** in the **New Project Wizard: Introduction** (The introduction does not appear if you previously turned it off)
3. On the **Directory, Name, Top-Level Entity** page, enter the following information:
   a. The working directory shown is correct. You do not have to change it.
   b. For the project name, browse to the synthesis directory that includes your Qsys project, **<*working_dir*>/pcie_de_gen1_x4_ast64/synthesis**. Select your variant name, **pcie_de_gen1_x4_ast64.v** . Then, click **Open.**
   c. If the top-level design entity and Qsys system names are identical, the Quartus II software treats the Qsys system as the top-level design entity.
4. Click **Next** to display the **Add Files** page.
5. Complete the following steps to add the Quartus II IP File (**.qip**)to the project:
   a. Click the **browse** button. The **Select File** dialog box appears.
   b. In the **Files of type** list, select **IP Variation Files (*.qip)**.
   c. Browse to the **<*working_dir*>/pcie_de_gen1_x4_ast64/synthesis** directory.
   d. Click **pcie_de_gen1_x4_ast64.qip** and then click **Open**.
   e. On the **Add Files** page, click **Add**, then click **OK**.
6. Click **Next** to display the **Device** page.
7. On the **Family & Device Settings** page, choose the following target device family and options:
   a. In the **Family** list, select Cyclone V (E/GX/GT/SX/SE/ST)
   b. In the **Devices** list, select Cyclone V **GX Extended Features.**.
   c. In the **Available Devices** list, select **5CGXFC7D6F31C7**.
8. Click **Next** to close this page and display the **EDA Tool Settings** page.
9. From the **Simulation** list, select **ModelSim®**. From the **Format** list, select the HDL language you intend to use for simulation.
10. Click **Next** to display the **Summary** page.
11. Check the **Summary** page to ensure that you have entered all the information correctly.
12. Click **Finish** to create the Quartus II project.
13. Add the Synopsys Design Constraint (SDC) commands shown in the following example to the top-level design file for your Quartus II project.
14. To compile your design using the Quartus II software, on the Processing menu, click **Start Compilation**. The Quartus II software then performs all the steps necessary to compile your design.
15. After compilation, expand the **TimeQuest Timing Analyzer** folder in the Compilation Report. Note whether the timing constraints are achieved in the Compilation Report.
16. If your design does not initially meet the timing constraints, you can find the optimal Fitter settings for your design by using the Design Space Explorer. To use the Design Space Explorer, click **Launch Design Space Explorer** on the tools menu.

Example 2-1: Synopsys Design Constraints

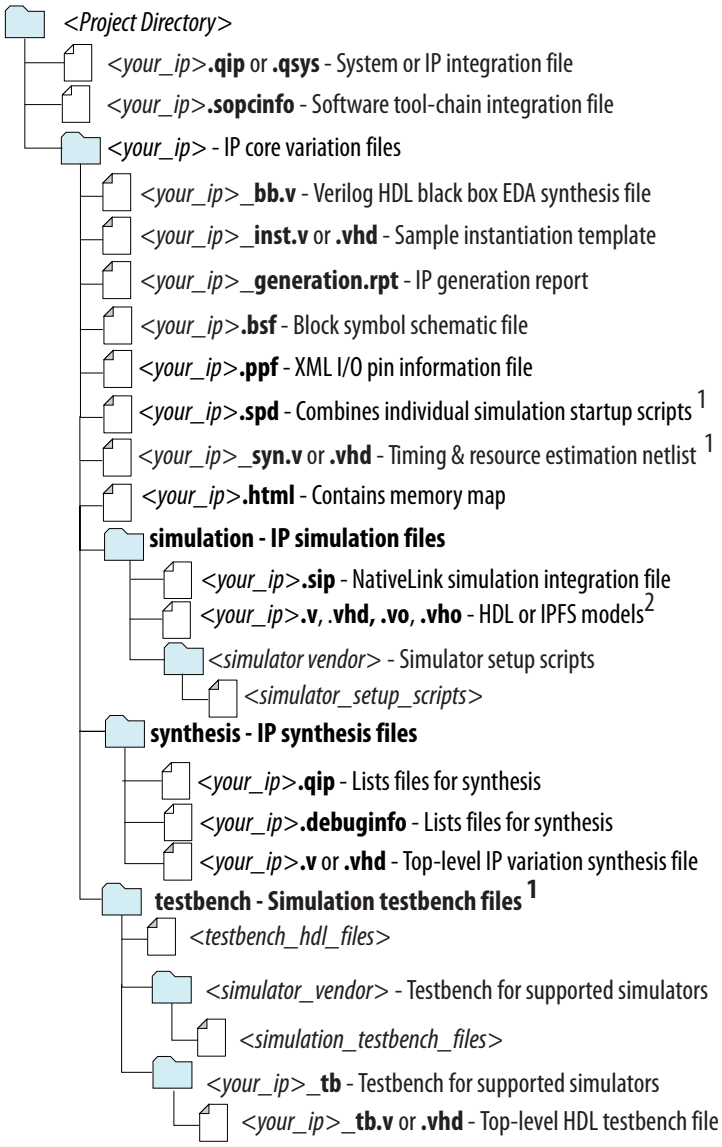
```
create_clock -period “100 MHz” -name {refclk_pci_express}{*refclk_*}
derive_pll_clocks
derive_clock_uncertainty
```

```
# PHY IP reconfig controller constraints
# Set reconfig_xcvr clock
# Modify to match the actual clock pin name
# used for this clock, and also changed to have the correct period set
create_clock -period "125 MHz" -name {reconfig_xcvr_clk}{*reconfig_xcvr_clk*}

# HIP Soft reset controller SDC constraints
set_false_path -to [get_registers* altpcie_rs_serdes|fifo_err_sync_r[0]]
set_false_path -from [get_registers *sv_xcvr_pipe_native*] -to[get_registers
*altpcie_rs_serdes|*]

# Hard IP testin pins SDC constraints
set_false_path -from [get_pins -compatibilitly_mode *hip_ctrl*]
```

Files Generated for Altera IP Cores

Figure 2-3: IP Core Generated Files

The Quartus II software generates the following output for your IP core.

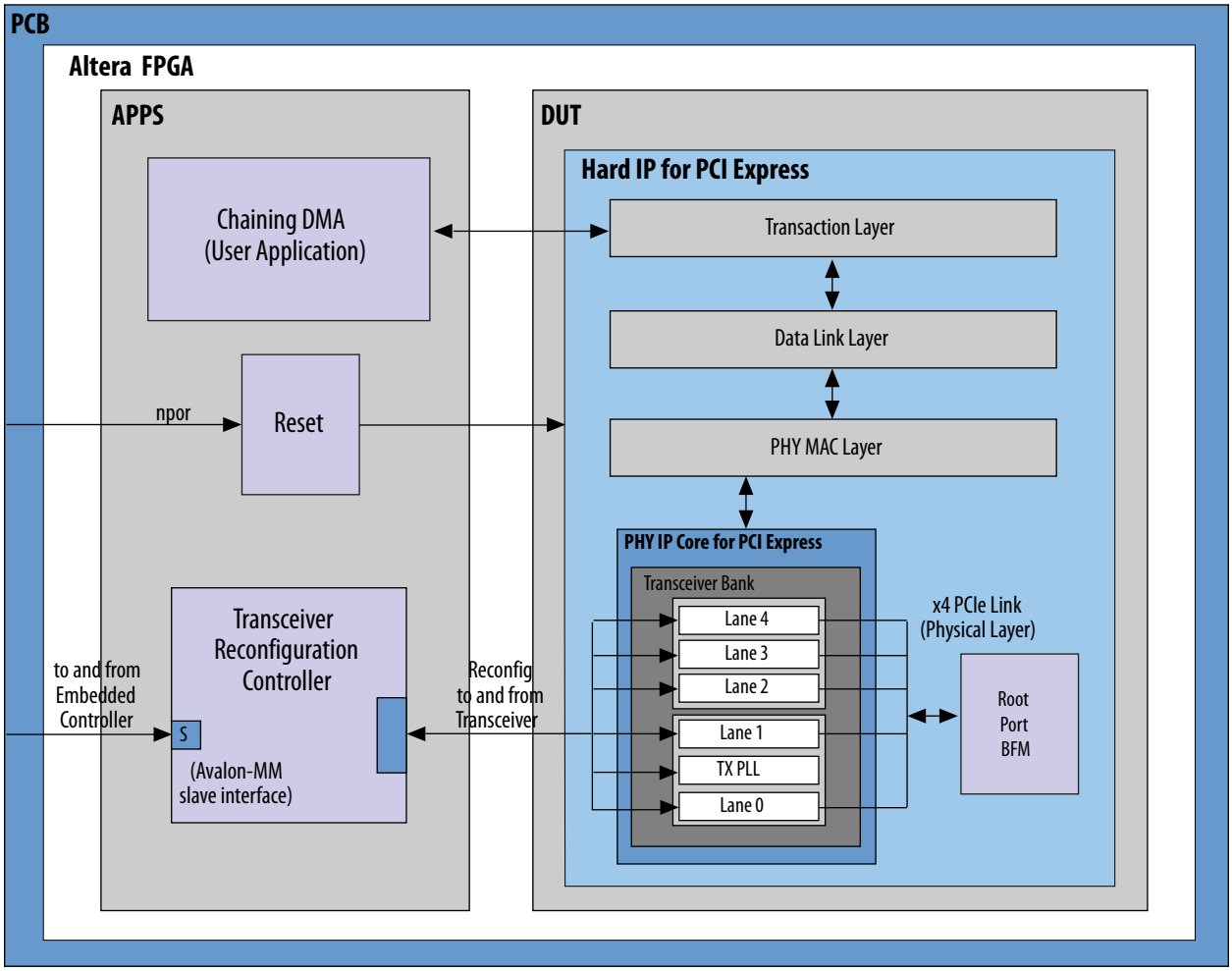


Notes:
1. If supported and enabled for your IP variation
2. If functional simulation models are generated

Modifying the Example Design

To use this example design as the basis of your own design, replace the Chaining DMA Example shown in the following figure with your own Application Layer design. Then modify the Root Port BFM driver to generate the transactions needed to test your Application Layer.

Figure 2-4: Testbench for PCI Express



Using the IP Catalog To Generate Your Cyclone V Hard IP for PCI Express as a Separate Component

You can also instantiate the Cyclone V Hard IP for PCI Express IP Core as a separate component for integration into your project.

You can use the Quartus II IP Catalog and IP Parameter Editor to select, customize, and generate files representing your custom IP variation. The IP Catalog (**Tools** > **IP Catalog**) automatically displays IP cores available for your target device. Double-click any IP core name to launch the parameter editor and generate files representing your IP variation.

For more information about the customizing and generating IP Cores refer to *Specifying IP Core Parameters and Options* in *Introduction to Altera IP Cores*. For more information about upgrading older IP cores to the current release, refer to *Upgrading Outdated IP Cores* in *Introduction to Altera IP Cores*.

Note: Your design must include the Transceiver Reconfiguration Controller IP Core and the Altera PCIe Reconfig Driver. Refer to the figure in the *Qsys Design Flow* section to learn how to connect this components.

Related Information

- **Introduction to Altera IP Cores**
- **Managing Quartus II Projects**

Parameter Settings 3

2014.12.15

UG-01110_avst

Subscribe Send Feedback

Avalon-ST System Settings

Table 3-1: System Settings for PCI Express

Parameter	Value	Description
Number of Lanes	**×1, ×2, ×4**	Specifies the maximum number of lanes supported.
Lane Rate	**Gen1 (2.5 Gbps)** **Gen2 (2.5/5.0 Gbps)**	Specifies the maximum data rate at which the link can operate.
Port type	**Root Port** **Native Endpoint** **Legacy Endpoint**	Specifies the port type. Altera recommends **Native Endpoint** for all new Endpoint designs. Select **Legacy Endpoint** only when you require I/O transaction support for compatibility. The **Legacy Endpoint** is not available for the Avalon-MM Cyclone V Hard IP for PCI Express. The Endpoint stores parameters in the Type 0 Configuration Space. The Root Port stores parameters in the Type 1 Configuration Space.
Application Interface	**Avalon-ST 64-bit** **Avalon-ST 128-bit**	Specifies the width of the Avalon-ST interface between the Application and Transaction Layers. The following widths are required:

Data Rate	Link Width	Interface Width
Gen1	×1	64 bits
	×2	64 bits
	×4	64 bits

© 2014 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

<table>
<tr><th>Parameter</th><th>Value</th><th>Description</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><th>Data Rate</th><th>Link Width</th><th>Interface Width</th></tr>
<tr><td rowspan="3">Gen2</td><td>×1</td><td>64 bits</td></tr>
<tr><td>×2</td><td>64 bits</td></tr>
<tr><td>×4</td><td>128 bits</td></tr>
</table>
</td></tr>
<tr><td>RX Buffer credit allocation - performance for received requests</td><td>Minimum
Low
Balanced
High
Maximum</td><td>Determines the allocation of posted header credits, posted data credits, non-posted header credits, completion header credits, and completion data credits in the 16 KByte RX buffer. The 5 settings allow you to adjust the credit allocation to optimize your system. The credit allocation for the selected setting displays in the message pane.

Refer to the Throughput Optimization chapter for more information about optimizing performance. The Flow Control chapter explains how the RX credit allocation and the Maximum payload RX Buffer credit allocation and the Maximum payload size that you choose affect the allocation of flow control credits. You can set the Maximum payload size parameter on the Device tab.

The Message window of the GUI dynamically updates the number of credits for Posted, Non-Posted Headers and Data, and Completion Headers and Data as you change this selection.</td></tr>
</table>

Parameter	Value	Description
		• **Minimum RX Buffer credit allocation -performance for received requests**–This setting configures the minimum PCIe specification allowed for non-posted and posted request credits, leaving most of the RX Buffer space for received completion header and data. Select this option for variations where application logic generates many read requests and only infrequently receives single requests from the PCIe link. • **Low**–This setting configures a slightly larger amount of RX Buffer space for non-posted and posted request credits, but still dedicates most of the space for received completion header and data. Select this option for variations where application logic generates many read requests and infrequently receives small bursts of requests from the PCIe link. This option is recommended for typical endpoint applications where most of the PCIe traffic is generated by a DMA engine that is located in the endpoint application layer logic. • **Balanced**–This setting allocates approximately half the RX Buffer space to received requests and the other half of the RX Buffer space to received completions. Select this option for variations where the received requests and received completions are roughly equal. • **High**–This setting configures most of the RX Buffer space for received requests and allocates a slightly larger than minimum amount of space for received completions. Select this option where most of the PCIe requests are generated by the other end of the PCIe link and the local application layer logic only infrequently generates a small burst of read requests. This option is recommended for typical root port applications where most of the PCIe traffic is generated by DMA engines located in the endpoints. • **Maximum**–This setting configures the minimum PCIe specification allowed amount of completion space, leaving most of the RX Buffer space for received requests. Select this option when most of the PCIe requests are generated by the other end of the PCIe link and the local application layer logic never or only infrequently generates single read requests. This option is recommended for control and status endpoint applications that don't generate any PCIe requests of their own and only are the target of write and read requests from the root complex.

Parameter	Value	Description
Reference clock frequency	**100 MHz** **125 MHz**	The *PCI Express Base Specification* requires a 100 MHz ±300 ppm reference clock. The 125 MHz reference clock is provided as a convenience for systems that include a 125 MHz clock source.
Use 62.5 MHz application clock	**On/Off**	This mode is only available only for Gen1 ×1.
Use deprecated RX Avalon-ST data byte enable port (rx_st_be)	**On/Off**	This parameter is only available for the Avalon-ST Cyclone V Hard IP for PCI Express.
Enable configuration via PCIe link	**On/Off**	When **On**, the Quartus II software places the Endpoint in the location required for configuration via protocol (CvP). For more information about CvP, click the *Configuration via Protocol (CvP)* link below.
Enable Hard IP Reconfiguration	**On/Off**	When **On**, you can use the Hard IP reconfiguration bus to dynamically reconfigure Hard IP read-only registers. For more information refer to *Hard IP Reconfiguration Interface*. This parameter is not available for the Avalon-MM IP Cores.
Number of Functions	1–8	Specifies the number of functions that share the same link.

Related Information

PCI Express Base Specification 2.1 or 3.0

Link Capabilities

Table 3-2: Link Capabilities

Parameter	Value	Description
Link port number	**0x01**	Sets the read-only value of the port number field in the `Link Capabilities Register`.
Slot clock configuration	**On/Off**	When **On**, indicates that the Endpoint or Root Port uses the same physical reference clock that the system provides on the connector. When **Off**, the IP core uses an independent clock regardless of the presence of a reference clock on the connector.

Port Function Parameters Shared Across All Port Functions

Device Capabilities

Table 3-3: Capabilities Registers

Parameter	Possible Values	Default Value	Description
Maximum payload size	**128 bytes** **256 bytes** **512 bytes**	128 bytes	Specifies the maximum payload size supported. This parameter sets the read-only value of the max payload size supported field of the Device Capabilities register (0x084[2:0]). Address: 0x084**.**
Number of tags supported per function	**32** **64**	32 - Avalon-ST	Indicates the number of tags supported for non-posted requests transmitted by the Application Layer. This parameter sets the values in the Device Control register (0x088) of the PCI Express capability structure described in Table 9–9 on page 9–5. The Transaction Layer tracks all outstanding completions for non-posted requests made by the Application Layer. This parameter configure**Tags supported**es the Transaction Layer for the maximum number to track. The Application Layer must set the tag values in all non-posted PCI Express headers to be less than this value. Values greater than 32 also set the extended tag field supported bit in the Configuration Space Device Capabilities register. The Application Layer can only use tag numbers greater than 31 if configuration software sets the Extended Tag Field Enable bit of the Device Control register. This bit is available to the Application Layer on the `tl_cfg_ctl` output signal as `cfg_devcsr[8]`.
Completion timeout range	**ABCD** **BCD** **ABC** **AB** **B** **A** **None**	ABCD	Indicates device function support for the optional completion timeout programmability mechanism. This mechanism allows system software to modify the completion timeout value. This field is applicable only to Root Ports and Endpoints that issue requests on their own behalf. Completion timeouts are specified and enabled in the Device Control 2 register (0x0A8) of the *PCI Express Capability Structure Version*. For all other

Parameter	Possible Values	Default Value	Description
			functions this field is reserved and must be hardwired to 0x0000b. Four time value ranges are defined: • Range A: 50 us to 10 ms • Range B: 10 ms to 250 ms • Range C: 250 ms to 4 s • Range D: 4 s to 64 s Bits are set to show timeout value ranges supported. The function must implement a timeout value in the range 50 s to 50 ms. The following values specify the range: • None – Completion timeout programming is not supported • 0001 Range A • 0010 Range B • 0011 Ranges A and B • 0110 Ranges B and C • 0111 Ranges A, B, and C • 1110 Ranges B, C and D • 1111 Ranges A, B, C, and D All other values are reserved. Altera recommends that the completion timeout mechanism expire in no less than 10 ms.
Implement completion timeout disable	**On/Off**	On	For Endpoints using PCI Express version 2.1 or 3.0, this option must be **On**. The timeout range is selectable. When **On**, the core supports the completion timeout disable mechanism via the PCI Express `Device Control Register 2`. The Application Layer logic must implement the actual completion timeout mechanism for the required ranges.

Error Reporting

Table 3-4: Error Reporting

Parameter	Value	Default Value	Description
Advanced error reporting (AER)	**On/Off**	Off	When **On**, enables the Advanced Error Reporting (AER) capability.

Parameter	Value	Default Value	Description
ECRC checking	**On/Off**	Off	When **On**, enables ECRC checking. Sets the read-only value of the ECRC check capable bit in the `Advanced Error Capabilities and Control Register`. This parameter requires you to enable the AER capability.
ECRC generation	**On/Off**	Off	When **On**, enables ECRC generation capability. Sets the read-only value of the ECRC generation capable bit in the `Advanced Error Capabilities and Control Register`. This parameter requires you to enable the AER capability. Not applicable for Avalon-MM DMA.
ECRC forwarding	**On/Off**	Off	When **On**, enables ECRC forwarding to the Application Layer. On the Avalon-ST RX path, the incoming TLP contains the ECRC dword (3) and the `TD` bit is set if an ECRC exists. On the transmit the TLP from the Application Layer must contain the ECRC dword and have the `TD` bit set. Not applicable for Avalon-MM DMA.

Link Capabilities

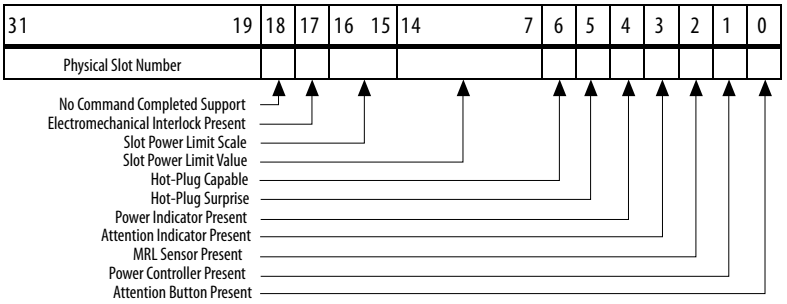
Table 3-5: Link Capabilities

Parameter	Value	Description
Link port number	**0x01**	Sets the read-only value of the port number field in the `Link Capabilities Register`.
Slot clock configuration	**On/Off**	When **On**, indicates that the Endpoint or Root Port uses the same physical reference clock that the system provides on the connector. When **Off**, the IP core uses an independent clock regardless of the presence of a reference clock on the connector.

(3) Throughout this user guide, the terms word, dword and qword have the same meaning that they have in the *PCI Express Base Specification*. A word is 16 bits, a dword is 32 bits, and a qword is 64 bits.

Slot Capabilities

Table 3-6: Slot Capabilities

Parameter	Value	Description
Use Slot register	**On/Off**	The slot capability is required for Root Ports if a slot is implemented on the port. Slot status is recorded in the `PCI Express Capabilities` register. This parameter is only supported in Root Port mode. Defines the characteristics of the slot. You turn on this option by selecting **Enable slot capability**. The various bits are defined as follows: 
Slot power scale	`0–3`	Specifies the scale used for the **Slot power limit**. The following coefficients are defined: • 0 = 1.0x • 1 = 0.1x • 2 = 0.01x • 3 = 0.001x The default value prior to hardware and firmware initialization is b'00. Writes to this register also cause the port to send the `Set_Slot_Power_Limit` Message. Refer to Section 6.9 of the *PCI Express Base Specification Revision* for more information.
Slot power limit	`0–255`	In combination with the **Slot power scale value**, specifies the upper limit in watts on power supplied by the slot. Refer to Section 7.8.9 of the *PCI Express Base Specification* for more information.
Slot number	`0-8191`	Specifies the slot number.

Related Information

PCI Express Base Specification Revision 2.1 or 3.0

Power Management

Table 3-7: Power Management Parameters

Parameter	Value	Description
Endpoint L0s acceptable latency	**Maximum of 64 ns** **Maximum of 128 ns** **Maximum of 256 ns** **Maximum of 512 ns** **Maximum of 1 us** **Maximum of 2 us** **Maximum of 4 us** **No limit**	This design parameter specifies the maximum acceptable latency that the device can tolerate to exit the L0s state for any links between the device and the root complex. It sets the read-only value of the Endpoint L0s acceptable latency field of the `Device Capabilities Register` (0x084). This Endpoint does not support the L0s or L1 states. However, in a switched system there may be links connected to switches that have L0s and L1 enabled. This parameter is set to allow system configuration software to read the acceptable latencies for all devices in the system and the exit latencies for each link to determine which links can enable Active State Power Management (ASPM). This setting is disabled for Root Ports. The default value of this parameter is 64 ns. This is the safest setting for most designs.
Endpoint L1 acceptable latency	**Maximum of 1 us** **Maximum of 2 us** **Maximum of 4 us** **Maximum of 8 us** **Maximum of 16 us** **Maximum of 32 us** **No limit**	This value indicates the acceptable latency that an Endpoint can withstand in the transition from the L1 to L0 state. It is an indirect measure of the Endpoint's internal buffering. It sets the read-only value of the Endpoint L1 acceptable latency field of the `Device Capabilities Register`. This Endpoint does not support the L0s or L1 states. However, a switched system may include links connected to switches that have L0s and L1 enabled. This parameter is set to allow system configuration software to read the acceptable latencies for all devices in the system and the exit latencies for each link to determine which links can enable Active State Power Management (ASPM). This setting is disabled for Root Ports. The default value of this parameter is 1 µs. This is the safest setting for most designs.

Port Function Parameters Defined Separately for All Port Functions

Base Address Register (BAR) and Expansion ROM Settings

The type and size of BARs available depend on port type.

Table 3-8: BAR Registers

Parameter	Value	Description
Type	**Disabled** **64-bit prefetchable memory** **32-bit non-prefetchable memory** **32-bit prefetchable memory** **I/O address space**	If you select 64-bit prefetchable memory, 2 contiguous BARs are combined to form a 64-bit prefetchable BAR; you must set the higher numbered BAR to **Disabled**. A non-prefetchable 64-bit BAR is not supported because in a typical system, the Root Port Type 1 Configuration Space sets the maximum non-prefetchable memory window to 32 bits. The BARs can also be configured as separate 32-bit memories. Defining memory as prefetchable allows contiguous data to be fetched ahead. Prefetching memory is advantageous when the requestor may require more data from the same region than was originally requested. If you specify that a memory is prefetchable, it must have the following 2 attributes: • Reads do not have side effects such as changing the value of the data read • Write merging is allowed The **32-bit prefetchable memory** and **I/O address space** BARs are only available for the **Legacy Endpoint**.
Size	**16 Bytes–8 EBytes**	Supports the following memory sizes: • 128 bytes–2 GBytes or 8 EBytes: **Endpoint** and **Root Port** variants • 6 bytes–4 KBytes: **Legacy Endpoint** variants
Expansion ROM	**Disabled–16 MBytes**	Specifies the size of the optional ROM. The expansion ROM is only available for the Avalon-ST interface.

Base and Limit Registers for Root Ports

Table 3-9: Base and Limit Registers for Function 0

The following table describes the `Base` and `Limit` registers which are available in the Type 1 Configuration Space for Root Ports. These registers are used for TLP routing and specify the address ranges assigned to components that are downstream of the Root Port or bridge.

Parameter	Value	Description
Input/ Output	**Disabled** **16-bit I/O addressing** **32-bit I/O addressing**	Specifies the address widths for the `IO base` and `IO limit` registers.
Prefetchable memory	**Disabled** **16-bit memory addressing** **32-bit memory addressing**	Specifies the address widths for the `Prefetchable Memory Base` register and `Prefetchable Memory Limit` register.

Related Information

PCI to PCI Bridge Architecture Specification

Device Identification Registers for Function *<n>*

Table 3-10: Device ID Registers

The following table lists the default values of the read-only Device ID registers. You can use the parameter editor to change the values of these registers. Refer to *Type 0 Configuration Space Registers* for the layout of the Device Identification registers.

Register Name	Range	Default Value	Description
Vendor ID	16 bits	0x00000000	Sets the read-only value of the `Vendor ID` register. This parameter cannot be set to 0xFFFF, per the *PCI Express Specification*. Address offset: 0x000.
Device ID	16 bits	0x00000001	Sets the read-only value of the `Device ID` register. This register is only valid in the Type 0 (Endpoint) Configuration Space. Address offset: 0x000.
Revision ID	8 bits	0x00000001	Sets the read-only value of the `Revision ID` register. Address offset: 0x008.

Register Name	Range	Default Value	Description
Class code	24 bits	0x00000000	Sets the read-only value of the `Class Code` register. Address offset: 0x008.
Subsystem Vendor ID	16 bits	0x00000000	Sets the read-only value of the `Subsystem Vendor ID` register in the PCI Type 0 Configuration Space. This parameter cannot be set to 0xFFFF per the *PCI Express Base Specification*. This value is assigned by PCI-SIG to the device manufacturer. This register is only valid in the Type 0 (Endpoint) Configuration Space. Address offset: 0x02C.
Subsystem Device ID	16 bits	0x00000000	Sets the read-only value of the `Subsystem Device ID` register in the PCI Type 0 Configuration Space. Address offset: 0x02C

At run time, you can change the values of these registers using the optional reconfiguration block signals.

Related Information

PCI Express Base Specification 2.1 or 3.0

Func <n> Device

Table 3-11: Func <n> Device

Parameter	Value	Description
Function Level Reset (FLR)	**On/Off**	Turn On this option to set the Function Level Reset Capability bit in the Device Capabilities register. This parameter applies to Endpoints only.

Func <n> MSI and MSI-X Capabilities

Table 3-12: Func <n> MSI and MSI-X Capabilities

Parameter	Value	Description
MSI messages requested	**1, 2, 4, 8, 16, 32**	Specifies the number of messages the Application Layer can request. Sets the value of the `Multiple Message Capable` field of the `Message Control` register, 0x050[31:16].
	MSI-X Capabilities	
Implement MSI-X	**On/Off**	When **On**, enables the MSI-X functionality.

Parameter	Value	Description
	Bit Range	
Table size	[10:0]	System software reads this field to determine the MSI-X Table size <*n*>, which is encoded as <*n–1*>. For example, a returned value of 2047 indicates a table size of 2048. This field is read-only. Legal range is 0–2047 (2^{11}). Address offset: 0x068[26:16]
Table Offset	[31:0]	Points to the base of the MSI-X Table. The lower 3 bits of the table BAR indicator (BIR) are set to zero by software to form a 32-bit qword-aligned offset [(4)]. This field is read-only.
Table BAR Indicator	[2:0]	Specifies which one of a function's BARs, located beginning at 0x10 in Configuration Space, is used to map the MSI-X table into memory space. This field is read-only. Legal range is 0–5.
Pending Bit Array (PBA) Offset	[31:0]	Used as an offset from the address contained in one of the function's Base Address registers to point to the base of the MSI-X PBA. The lower 3 bits of the PBA BIR are set to zero by software to form a 32-bit qword-aligned offset. This field is read-only.
PBA BAR Indicator	[2:0]	Specifies the function Base Address registers, located beginning at 0x10 in Configuration Space, that maps the MSI-X PBA into memory space. This field is read-only. Legal range is 0–5.

Related Information

PCI Express Base Specification Revision 2.1 or 3.0

(4) Throughout this user guide, the terms word, dword and qword have the same meaning that they have in the *PCI Express Base Specification*. A word is 16 bits, a dword is 32 bits, and a qword is 64 bits.

Func <n> Legacy Interrupt

Table 3-13: Func <n> Legacy Interrupt

Parameter	Value	Description
Legacy Interrupt (INTx)	**INTA** **INTB** **INTC** **INTD** **None**	When selected, allows you to drive legacy interrupts to the Application Layer.

Interfaces and Signal Descriptions

4

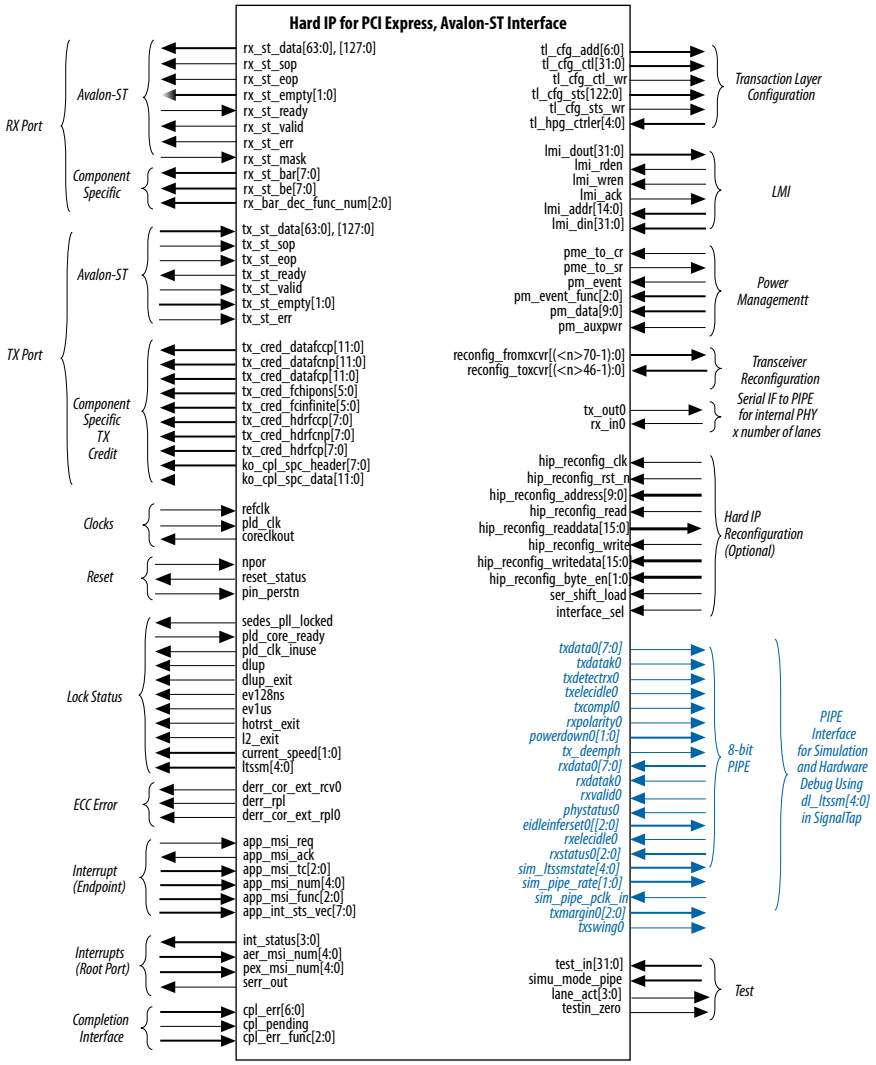
2014.12.15

UG-01110_avst

Subscribe

Send Feedback

Figure 4-1: Avalon-ST Hard IP for PCI Express Top-Level Signals



© 2014 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

Avalon-ST RX Interface

Table 4-1: 64- or 128-Bit Avalon-ST RX Datapath

The RX data signal can be 64 or 128 bits.

Signal	Direction	Description
rx_st_data[<*n*>-1:0]	Output	Receive data bus. Refer to figures following this table for the mapping of the Transaction Layer's TLP information to rx_st_data and examples of the timing of this interface. Note that the position of the first payload dword depends on whether the TLP address is qword aligned. The mapping of message TLPs is the same as the mapping of TLPs with 4-dword headers. When using a 64-bit Avalon-ST bus, the width of rx_st_data is 64. When using a 128-bit Avalon-ST bus, the width of rx_st_data is 128.
rx_st_sop	Output	Indicates that this is the first cycle of the TLP when rx_st_valid is asserted.
rx_st_eop	Output	Indicates that this is the last cycle of the TLP when rx_st_valid is asserted.

rx_st_empty	Output	Indicates the number of empty qwords in rx_st_data. Not used when rx_st_data is 64 bits. Valid only when rx_st_eop is asserted in 128-bit mode.
		For 128-bit data, only bit 0 applies; this bit indicates whether the upper qword contains data.
		• 128-Bit interface:
		  • rx_st_empty = 0, rx_st_data[127:0]contains valid data
		  • rx_st_empty = 1, rx_st_data[63:0] contains valid data

rx_st_ready	Input	Indicates that the Application Layer is ready to accept data. The Application Layer deasserts this signal to throttle the data stream. If rx_st_ready is asserted by the Application Layer on cycle <*n*>, then <*n* + > readyLatency > is a ready cycle, during which the Transaction Layer may assert valid and transfer data. The RX interface supports a readyLatency of 2 cycles.
rx_st_valid	Output	Clocks rx_st_data into the Application Layer. Deasserts within 2 clocks of rx_st_ready deassertion and reasserts within 2 clocks of rx_st_ready assertion if more data is available to send.

Signal	Direction	Description
rx_st_err	Output	Indicates that there is an uncorrectable error correction coding (ECC) error in the internal RX buffer. Active when ECC is enabled. ECC is automatically enabled by the Quartus II assembler. ECC corrects single-bit errors and detects double-bit errors on a per byte basis. When an uncorrectable ECC error is detected, rx_st_err is asserted for at least 1 cycle while rx_st_valid is asserted. Altera recommends resetting the Cyclone V Hard IP for PCI Express when an uncorrectable double-bit ECC error is detected.

Related Information

Avalon Interface Specifications.

Avalon-ST RX Component Specific Signals

Table 4-2: Avalon-ST RX Component Specific Signals

Signal	Direction	Description
rx_st_mask	Input	The Application Layer asserts this signal to tell the Hard IP to stop sending non-posted requests. This signal can be asserted at any time. The total number of non-posted requests that can be transferred to the Application Layer after rx_st_mask is asserted is not more than 10.

Signal	Direction	Description
rx_st_bar[7:0]	Output	The decoded BAR bits for the TLP. Valid for MRd, MWr, IOWR, and IORD TLPs. Ignored for the completion or message TLPs. Valid during the cycle in which rx_st_sop is asserted. Refer to *64-Bit Avalon-ST rx_st_data<n> Cycle Definitions for 4-Dword Header TLPs with Non-Qword Addresses* and *128-Bit Avalon-ST rx_st_data<n> Cycle Definition for 3-Dword Header TLPs with Qword Aligned Addresses* for the timing of this signal for 64- and 128-bit data, respectively. The following encodings are defined for Endpoints: • Bit 0: BAR 0 • Bit 1: BAR 1 • Bit 2: Bar 2 • Bit 3: Bar 3 • Bit 4: Bar 4 • Bit 5: Bar 5 • Bit 6: Expansion ROM • Bit 7: Reserved The following encodings are defined for Root Ports: • Bit 0: BAR 0 • Bit 1: BAR 1 • Bit 2: Primary Bus number • Bit 3: Secondary Bus number • Bit 4: Secondary Bus number to Subordinate Bus number window • Bit 5: I/O window • Bit 6: Non-Prefetchable window • Bit 7: Prefetchable window

Signal	Direction	Description
rx_st_be[<*n*>-1:0]	Output	Byte enables corresponding to the rx_st_data. The byte enable signals only apply to PCI Express Memory Write and I/O Write TLP payload fields. When using 64-bit Avalon-ST bus, the width of rx_st_be is 8 bits. When using 128-bit Avalon-ST bus, the width of rx_st_be is 16 bits. This signal is optional. You can derive the same information by decoding the FBE and LBE fields in the TLP header. The byte enable bits correspond to data bytes as follows: • rx_st_data[127:120] = rx_st_be[15] • rx_st_data[119:112] = rx_st_be[14] • rx_st_data[111:104] = rx_st_be[13] • rx_st_data[95:88] = rx_st_be[12] • rx_st_data[87:80] = rx_st_be[11] • rx_st_data[79:72] = rx_st_be[10] • rx_st_data[71:64] = rx_st_be[9] • rx_st_data[7:0] = rx_st_be[8] • rx_st_data[63:56] = rx_st_be[7] • rx_st_data[55:48] = rx_st_be[6] • rx_st_data[47:40] = rx_st_be[5] • rx_st_data[39:32] = rx_st_be[4] • rx_st_data[31:24] = rx_st_be[3] • rx_st_data[23:16] = rx_st_be[2] • rx_st_data[15:8] = rx_st_be[1] • rx_st_data[7:0] = rx_st_be[0] This signal is deprecated.
rx_st_parity[<*n*>-1:0]	Output	The IP core generates byte parity when you turn on **Enable byte parity ports on Avalon-ST interface** on the **System Settings** tab of the parameter editor. Each bit represents odd parity of the associated byte of the rx_st_datarx_st_data bus. For example, bit[0] corresponds to rx_st_data[7:0] rx_st_data[7:0], bit[1] corresponds to rx_st_data[15:8].
rx_bar_dec_func_num[2:0]	Output	Specifies which function the rx_st_bar signal applies to.

For more information about the Avalon-ST protocol, refer to the *Avalon Interface Specifications*.

Related Information

Avalon Interface Specifications.

Data Alignment and Timing for the 64-Bit Avalon-ST RX Interface

To facilitate the interface to 64-bit memories, the Cyclone V Hard IP for PCI Express aligns data to the qword or 64 bits by default. Consequently, if the header presents an address that is not qword aligned, the Hard IP block shifts the data within the qword to achieve the correct alignment.

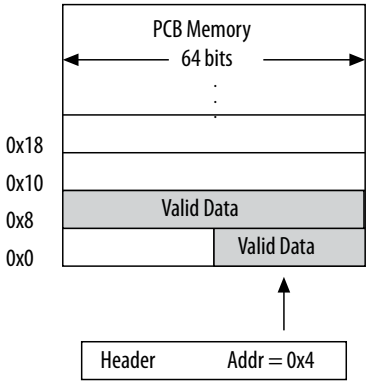
Qword alignment applies to all types of request TLPs with data, including the following TLPs:

- Memory writes
- Configuration writes
- I/O writes

The alignment of the request TLP depends on bit 2 of the request address. For completion TLPs with data, alignment depends on bit 2 of the `lower address` field. This bit is always 0 (aligned to qword boundary) for completion with data TLPs that are for configuration read or I/O read requests.

Figure 4-2: Qword Alignment

The following figure shows how an address that is not qword aligned, 0x4, is stored in memory. The byte enables only qualify data that is being written. This means that the byte enables are undefined for 0x0–0x3. This example corresponds to *64-Bit Avalon-ST rx_st_data<n> Cycle Definition for 3-Dword Header TLPs with Non-Qword Aligned Address*.



The following table shows the byte ordering for header and data packets.

Table 4-3: Mapping Avalon-ST Packets to PCI Express TLPs

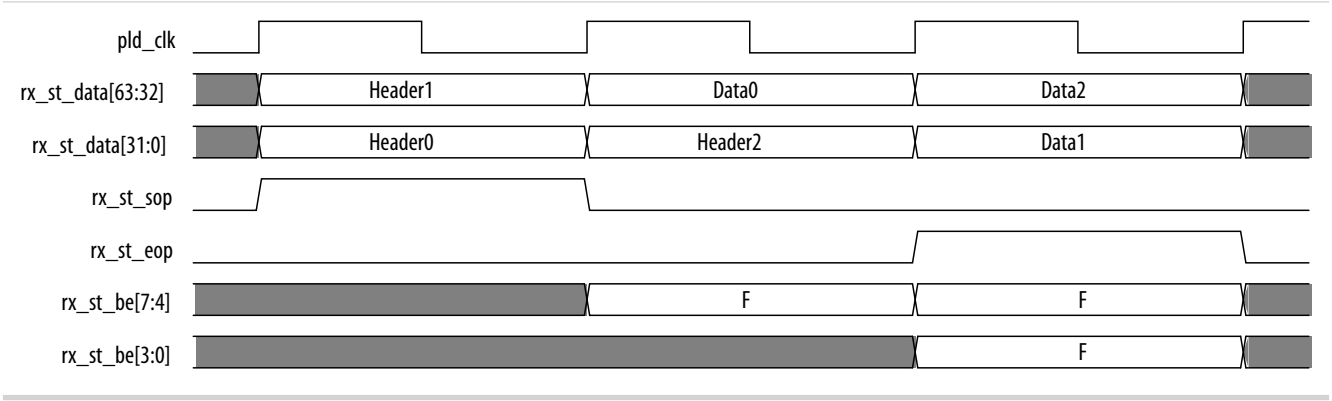
Packet	TLP
Header0	pcie_hdr_byte0, pcie_hdr _byte1, pcie_hdr _byte2, pcie_hdr _byte3
Header1	pcie_hdr _byte4, pcie_hdr _byte5, pcie_hdr byte6, pcie_hdr _byte7
Header2	pcie_hdr _byte8, pcie_hdr _byte9, pcie_hdr _byte10, pcie_hdr _byte11
Header3	pcie_hdr _byte12, pcie_hdr _byte13, header_byte14, pcie_hdr _byte15
Data0	pcie_data_byte3, pcie_data_byte2, pcie_data_byte1, pcie_data_byte0

Packet	TLP
Data1	pcie_data_byte7, pcie_data_byte6, pcie_data_byte5, pcie_data_byte4
Data2	pcie_data_byte11, pcie_data_byte10, pcie_data_byte9, pcie_data_byte8
Data<*n*>	pcie_data_byte<*4n+3*>, pcie_data_byte<*4n+2*>, pcie_data_byte<*4n+1*>, pcie_data_byte<*n*>

The following figure illustrates the mapping of Avalon-ST RX packets to PCI Express TLPs for a three dword header with non-qword aligned addresses with a 64-bit bus. In this example, the byte address is unaligned and ends with 0x4, causing the first data to correspond to `rx_st_data[63:32]` .

Note: The Avalon-ST protocol, as defined in *Avalon Interface Specifications*, is big endian, while the Hard IP for PCI Express packs symbols into words in little endian format. Consequently, you cannot use the standard data format adapters available in Qsys.

Figure 4-3: 64-Bit Avalon-ST rx_st_data<n> Cycle Definition for 3-Dword Header TLPs with Non-Qword Aligned Address



The following figure illustrates the mapping of Avalon-ST RX packets to PCI Express TLPs for a three dword header with qword aligned addresses. Note that the byte enables indicate the first byte of data is not valid and the last dword of data has a single valid byte.

Figure 4-4: 64-Bit Avalon-ST rx_st_data<n> Cycle Definition for 3-Dword Header TLPs with Qword Aligned Address

In the following figure, `rx_st_be[7:4]` corresponds to `rx_st_data[63:32]`. `rx_st_be[3:0]` corresponds to `rx_st_data[31:0]`.

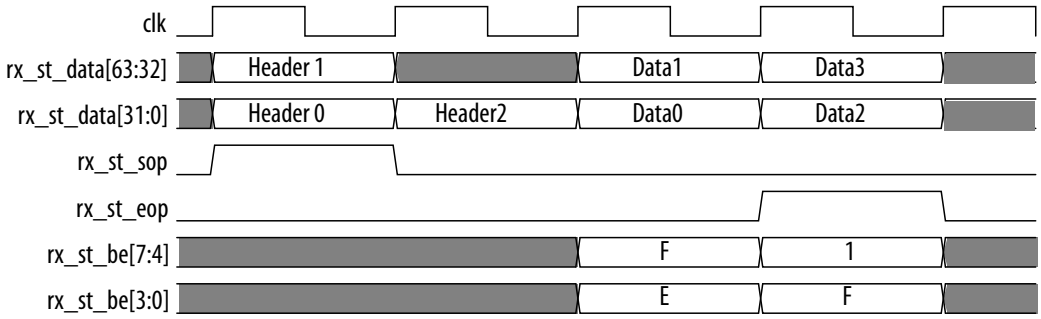


Figure 4-5: 64-Bit Application Layer Backpressures Transaction Layer

The following figure illustrates the timing of the RX interface when the Application Layer backpressures the Cyclone V Hard IP for PCI Express by deasserting `rx _st_ready`. The `rx_st_valid` signal deasserts within three cycles after `rx_st_ready` is deasserted. In this example, `rx_st_valid` is deasserted in the next cycle. `rx_st_data` is held until the Application Layer is able to accept it.

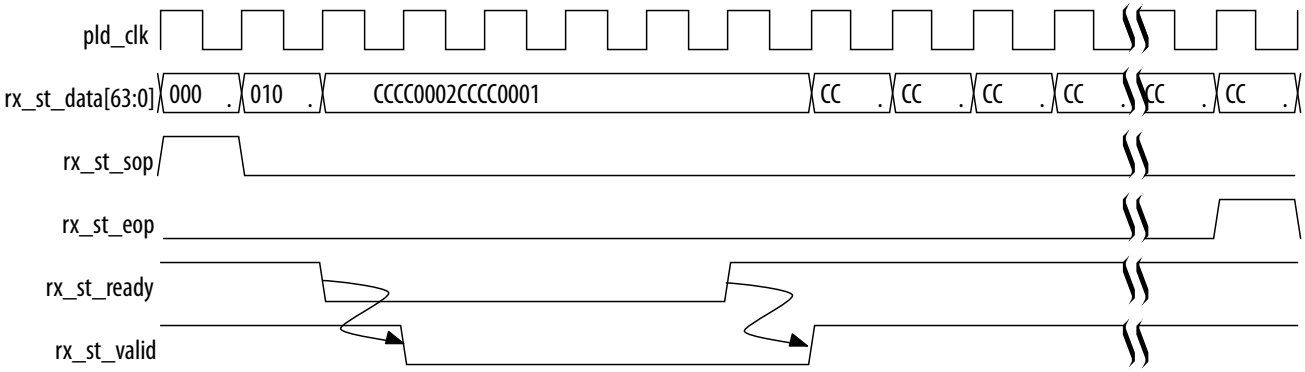
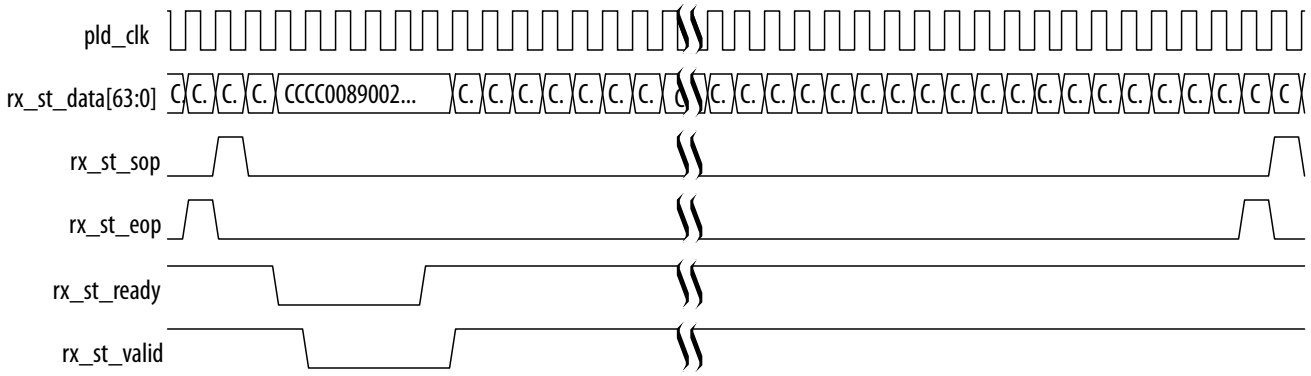


Figure 4-6: 4-Bit Avalon-ST Interface Back-to-Back Transmission

The following figure illustrates back-to-back transmission on the 64-bit Avalon-ST RX interface with no idle cycles between the assertion of `rx_st_eop` and `rx_st_sop`.



Related Information

- **Transaction Layer Packet (TLP) Header Formats** on page 18-1
- **Avalon Interface Specifications**

Data Alignment and Timing for the 128-Bit Avalon-ST RX Interface

Figure 4-7: 128-Bit Avalon-ST rx_st_data<n> Cycle Definition for 3-Dword Header TLPs with Qword Aligned Addresses

The following figure shows the mapping of 128-bit Avalon-ST RX packets to PCI Express TLPs for TLPs with a three dword header and qword aligned addresses. The assertion of `rx_st_empty` in a `rx_st_eop` cycle, indicates valid data on the lower 64 bits of `rx_st _data`.

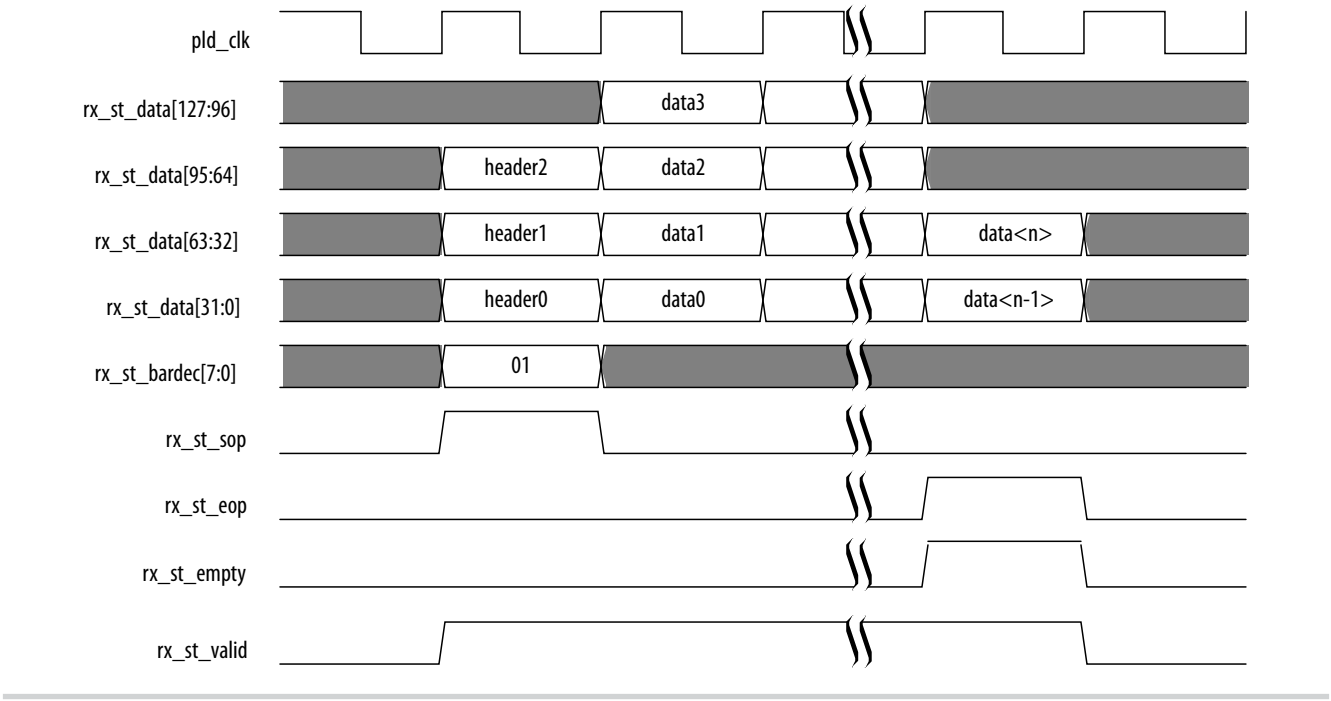


Figure 4-8: 128-Bit Avalon-ST rx_st_data<n> Cycle Definition for 3-Dword Header TLPs with non-Qword Aligned Addresses

The following figure shows the mapping of 128-bit Avalon-ST RX packets to PCI Express TLPs for TLPs with a 3 dword header and non-qword aligned addresses. In this case, bits[127:96] represent Data0 because address[2] in the TLP header is set. The assertion of `rx_st_empty` in a `rx_st_eop` cycle indicates valid data on the lower 64 bits of `rx_st_data`.

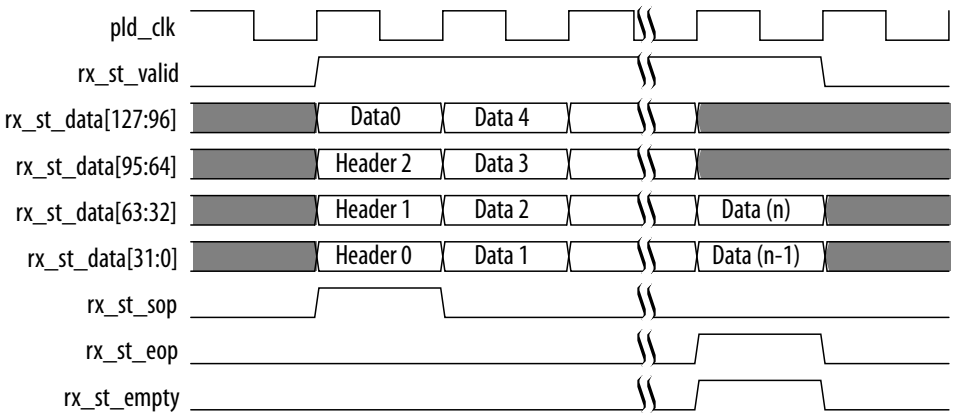


Figure 4-9: 128-Bit Avalon-ST rx_st_data Cycle Definition for 4-Dword Header TLPs with non-Qword Aligned Addresses

The following figure shows the mapping of 128-bit Avalon-ST RX packets to PCI Express TLPs for a four dword header with non-qword aligned addresses. In this example, `rx_st_empty` is low because the data is valid for all 128 bits in the `rx_st_eop` cycle.

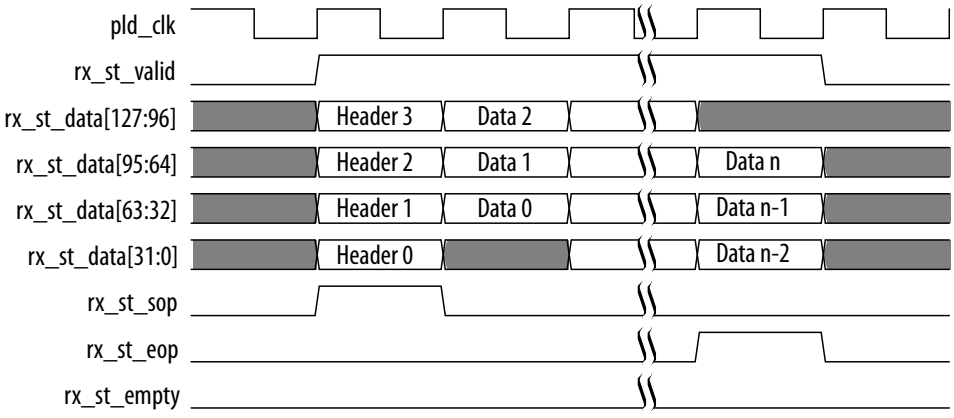


Figure 4-10: 128-Bit Avalon-ST rx_st_data Cycle Definition for 4-Dword Header TLPs with Qword Aligned Addresses

The following figure shows the mapping of 128-bit Avalon-ST RX packets to PCI Express TLPs for a four dword header with qword aligned addresses. In this example, `rx_st_empty` is low because data is valid for all 128-bits in the `rx_st_eop` cycle.

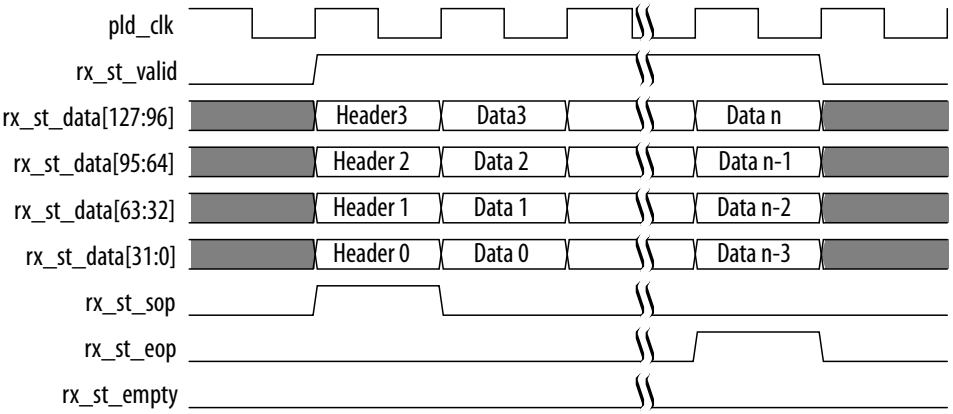
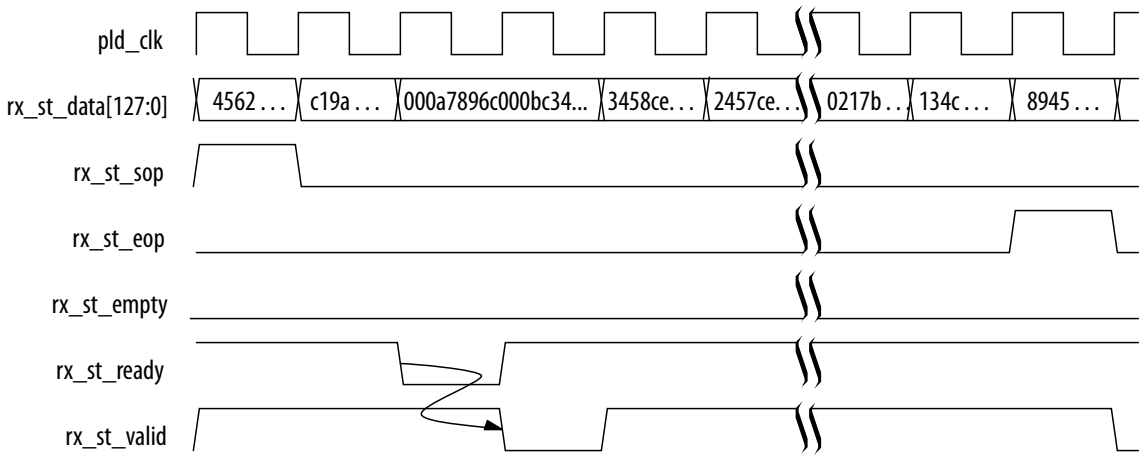


Figure 4-11: 128-Bit Application Layer Backpressures Hard IP Transaction Layer for RX Transactions

The following figure illustrates the timing of the RX interface when the Application Layer backpressures the Hard IP by deasserting `rx_st_ready`. The `rx_st_valid` signal deasserts within three cycles after `rx_st_ready` is deasserted. In this example, `rx_st_valid` is deasserted in the next cycle. `rx_st_data` is held until the Application Layer is able to accept it.



The following figure illustrates back-to-back transmission on the 128-bit Avalon-ST RX interface with no idle cycles between the assertion of `rx_st_eop` and `rx_st_sop`.

Figure 4-12: 128-Bit Avalon-ST Interface Back-to-Back Transmission

The following figure illustrates back-to-back transmission on the 128-bit Avalon-ST RX interface with no idle cycles between the assertion of `rx_st_eop` and `rx_st_sop`.

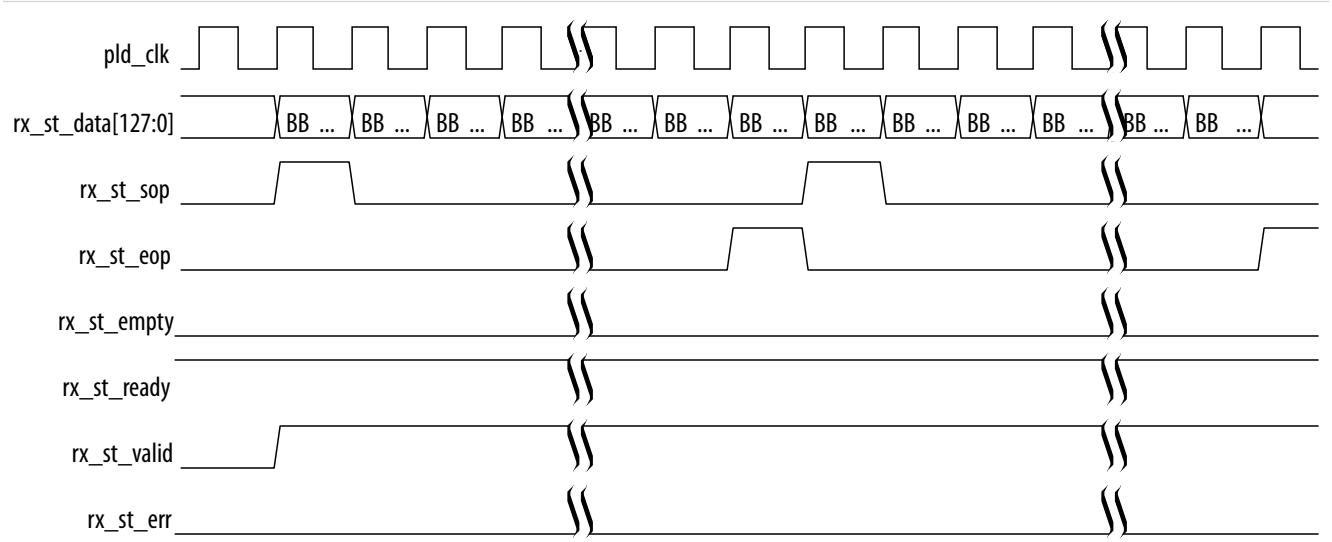
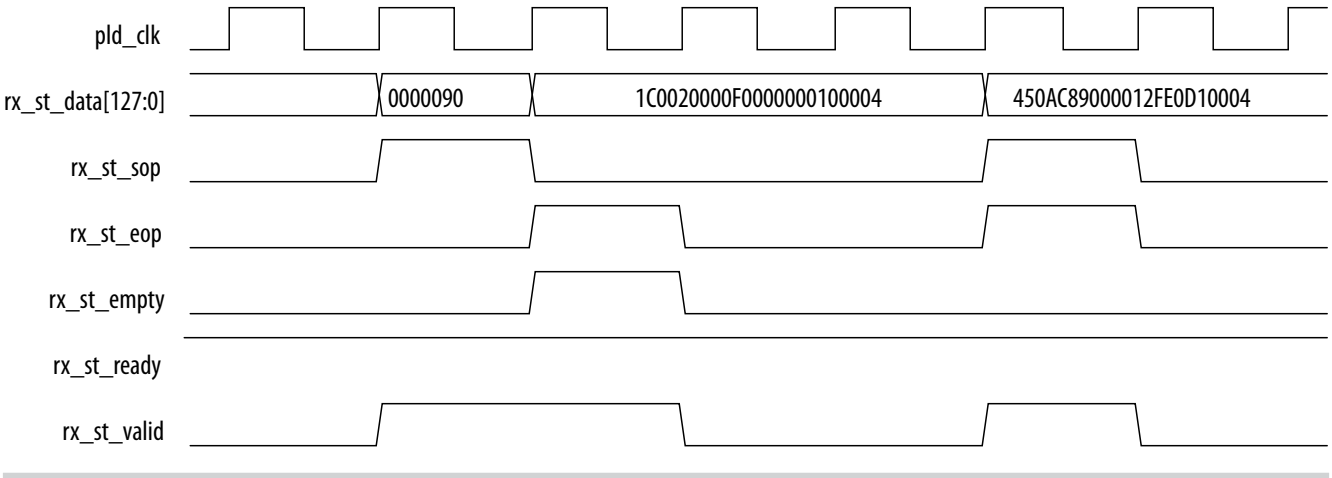


Figure 4-13: 128-Bit Packet Examples of rx_st_empty and Single-Cycle Packet

The following figure illustrates a two-cycle packet with valid data in the lower qword (`rx_st_data[63:0]`) and a one-cycle packet where the `rx_st_sop` and `rx_st_eop` occur in the same cycle.



For a complete description of the TLP packet header formats, refer to Appendix A, Transaction Layer Packet (TLP) Header Formats.

Avalon-ST TX Interface

The following table describes the signals that comprise the Avalon-ST TX Datapath. The TX data signal can be 64 or 128.

Table 4-4: 64- or 128-Bit Avalon-ST TX Datapath

Signal	Direction	Description
tx_st_data[<*n*>-1:0]	Input	Data for transmission. Transmit data bus. Refer to the following sections on data alignment for the 64- and 128-bit interfaces for the mapping of TLP packets to tx_st_data and examples of the timing of this interface. When using a 64-bit Avalon-ST bus, the width of tx_st_d ata is 64. When using a 128-bit Avalon-ST bus, the width of tx_st_data is 128 bits. The Application Layer must provide a properly formatted TLP on the TX interface. The mapping of message TLPs is the same as the mapping of Transaction Layer TLPs with 4 dword headers. The number of data cycles must be correct for the length and address fields in the header. Issuing a packet with an incorrect number of data cycles results in the TX interface hanging and becoming unable to accept further requests. <*n*> = 64 or 128.
tx_st_sop	Input	Indicates first cycle of a TLP when asserted together with tx_st_ valid.
tx_st_eop	Input	Indicates last cycle of a TLP when asserted together with tx_st_ valid.
tx_st_ready	Output	Indicates that the Transaction Layer is ready to accept data for transmission. The core deasserts this signal to throttle the data stream. tx_st_ready may be asserted during reset. The Application Layer should wait at least 2 clock cycles after the reset is released before issuing packets on the Avalon-ST TX interface. The reset_status signal can also be used to monitor when the IP core has come out of reset. If tx_st_ready is asserted by the Transaction Layer on cycle <*n*>, then <*n* + readyLatency> is a ready cycle, during which the Application Layer may assert valid and transfer data. When tx_st_ready, tx_st_valid and tx_st_data are registered (the typical case), Altera recommends a readyLatency of 2 cycles to facilitate timing closure; however, a readyLatency of 1 cycle is possible. If no other delays are added to the read-valid latency, the resulting delay corresponds to a readyLatency of 2.

Signal	Direction	Description
tx_st_valid	Input	Clocks tx_st_data to the core when tx_st_ready is also asserted. Between tx_st_sop and tx_st_eop, tx_st_valid must not be deasserted in the middle of a TLP except in response to tx_st_ready deassertion. When tx_st_ready deasserts, this signal must deassert within 1 or 2 clock cycles. When tx_st_ready reasserts, and tx_st_data is in mid-TLP, this signal must reassert within 2 cycles. The figure entitled*64-Bit Transaction Layer Backpressures the Application Layer* illustrates the timing of this signal. To facilitate timing closure, Altera recommends that you register both the tx_st_ready and tx_st_valid signals. If no other delays are added to the ready-valid latency, the resulting delay corresponds to a readyLatency of 2.
tx_st_empty[1:0]	Input	Indicates the number of qwords that are empty during cycles that contain the end of a packet. When asserted, the empty dwords are in the high-order bits. Valid only when tx_st_eop is asserted. Not used when tx_st_data is 64 bits. For 128-bit data, only bit 0 applies and indicates whether the upper qword contains data. For the 128-Bit interface: • If tx_st_empty = 0, tx_st_data[127:0] contains valid data. • If tx_st_empty = 1, tx_st_data[63:0] contains valid data.
tx_st_err	Input	Indicates an error on transmitted TLP. This signal is used to nullify a packet. It should only be applied to posted and completion TLPs with payload. To nullify a packet, assert this signal for 1 cycle after the SOP and before the EOP. When a packet is nullified, the following packet should not be transmitted until the next clock cycle. tx_st_err is not available for packets that are 1 or 2 cycles long. Refer to the figure entitled *128-Bit Avalon-ST tx_st_data Cycle Definition for 3-Dword Header TLP with non-Qword Aligned Address* for a timing diagram that illustrates the use of the error signal. Note that it must be asserted while the valid signal is asserted.
Component Specific Signals		
tx_cred_ datafccp[11:0]	Output	Data credit limit for the received FC completions. Each credit is 16 bytes.
tx_cred_ datafcnp[11:0]	Output	Data credit limit for the non-posted requests. Each credit is 16 bytes.

Signal	Direction	Description
tx_cred_datafcp[11:0]	Output	Data credit limit for the FC posted writes. Each credit is 16 bytes.
tx_cred_ fchipcons[5:0]	Output	Asserted for 1 cycle each time the Hard IP consumes a credit. These credits are from messages that the Hard IP for PCIe generates for the following reasons: • To respond to memory read requests • To send error messages This signal is not asserted when an Application Layer credit is consumed. The Application Layer must keep track of its own consumed credits. To calculate the total credits consumed, the Application Layer must add its own credits consumed to those consumed by the Hard IP for PCIe. The credit signals are valid after dlup (data link up) is asserted. The 6 bits of this vector correspond to the following 6 types of credit types: • [5]: posted headers • [4]: posted data • [3]: non-posted header • [2]: non-posted data • [1]: completion header • [0]: completion data During a single cycle, the IP core can consume either a single header credit or both a header and a data credit.
tx_cred_ fcinfinite[5:0]	Output	When asserted, indicates that the corresponding credit type has infinite credits available and does not need to calculate credit limits. The 6 bits of this vector correspond to the following 6 types of credit types: • [5]: posted headers • [4]: posted data • [3]: non-posted header • [2]: non-posted data • [1]: completion header • [0]: completion data
tx_cred_hdrfccp[7:0]	Output	Header credit limit for the FC completions. Each credit is 20 bytes.
tx_cred_hdrfcnp[7:0]	O	Header limit for the non-posted requests. Each credit is 20 bytes.

Signal	Direction	Description
tx_cred_hdrfcp[7:0]	O	Header credit limit for the FC posted writes. Each credit is 20 bytes.
ko_cpl_spc_ header[7:0]	Output	The Application Layer can use this signal to build circuitry to prevent RX buffer overflow for completion headers. Endpoints must advertise infinite space for completion headers; however, RX buffer space is finite. ko_cpl_spc_header is a static signal that indicates the total number of completion headers that can be stored in the RX buffer.
ko_cpl_spc_data[11:0]	Output	The Application Layer can use this signal to build circuitry to prevent RX buffer overflow for completion data. Endpoints must advertise infinite space for completion data; however, RX buffer space is finite. ko_cpl_spc_data is a static signal that reflects the total number of 16 byte completion data units that can be stored in the completion RX buffer.

Avalon-ST Packets to PCI Express TLPs

The following figures illustrate the mappings between Avalon-ST packets and PCI Express TLPs. These mappings apply to all types of TLPs, including posted, non-posted, and completion TLPs. Message TLPs use the mappings shown for four dword headers. TLP data is always address-aligned on the Avalon-ST interface whether or not the lower dwords of the header contains a valid address, as may be the case with TLP type (message request with data payload).

For additional information about TLP packet headers, refer to Appendix A, Transaction Layer Packet (TLP) Header Formats and *Section 2.2.1 Common Packet Header Fields* in the *PCI Express Base Specification* .

Related Information

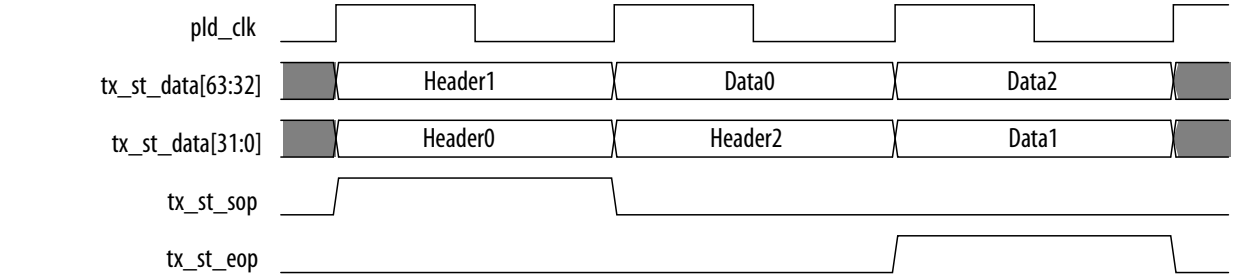
PCI Express Base Specification Revision 2.1 or 3.0

Data Alignment and Timing for the 64-Bit Avalon-ST TX Interface

Figure 4-14:

The following figure illustrates the mapping between Avalon-ST TX packets and PCI Express TLPs for three dword header TLPs with non-qword aligned addresses on a 64-bit bus.

Figure 4-15: 64-Bit Avalon-ST tx_st_data Cycle Definition for 3-Dword Header TLP with Non-Qword Aligned Address

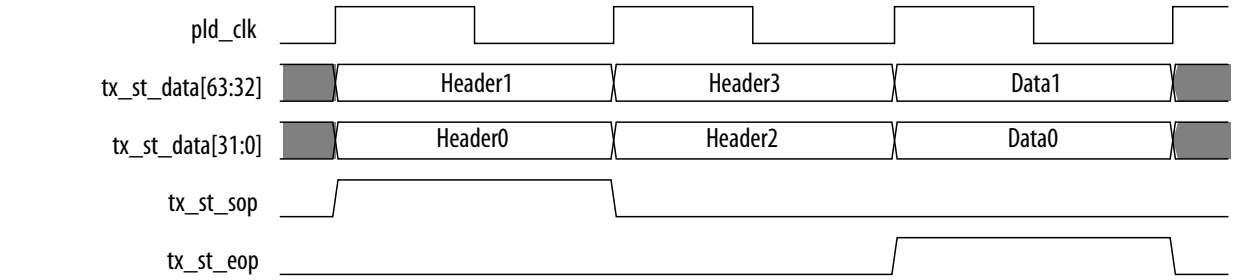


This figure illustrates the storage of non-qword aligned data.) Non-qword aligned address occur when `address[2]` is set. When address[2] is set, `tx_st_data[63:32]`contains `Data0` and `tx_st_data[31:0]` contains dword `header2`. In this figure, the headers are formed by the following bytes:

```
H0 ={pcie_hdr_byte0, pcie_hdr _byte1, pcie_hdr _byte2, pcie_hdr _byte3}
H1 = {pcie_hdr_byte4, pcie_hdr _byte5, header pcie_hdr byte6, pcie_hdr _byte7}
H2 = {pcie_hdr _byte8, pcie_hdr _byte9, pcie_hdr _byte10, pcie_hdr _byte11}
Data0 = {pcie_data_byte3, pcie_data_byte2, pcie_data_byte1, pcie_data_byte0}
Data1 = {pcie_data_byte7, pcie_data_byte6, pcie_data_byte5, pcie_data_byte4}
Data2 = {pcie_data_byte11, pcie_data_byte10, pcie_data_byte9, pcie_data_byte8}
```

The following figure illustrates the mapping between Avalon-ST TX packets and PCI Express TLPs for a four dword header with qword aligned addresses on a 64-bit bus

Figure 4-16: 64-Bit Avalon-ST tx_st_data Cycle Definition for 4-Dword TLP with Qword Aligned Address



In this figure, the headers are formed by the following bytes.

```
H0 = {pcie_hdr_byte0, pcie_hdr _byte1, pcie_hdr _byte2, pcie_hdr _byte3}
H1 = {pcie_hdr _byte4, pcie_hdr _byte5, pcie_hdr byte6, pcie_hdr _byte7}
H2 = {pcie_hdr _byte8, pcie_hdr _byte9, pcie_hdr _byte10, pcie_hdr _byte11}
H3 = pcie_hdr _byte12, pcie_hdr _byte13, header_byte14, pcie_hdr _byte15}, 4 dword
header only
```

```
Data0 = {pcie_data_byte3, pcie_data_byte2, pcie_data_byte1, pcie_data_byte0}
Data1 = {pcie_data_byte7, pcie_data_byte6, pcie_data_byte5, pcie_data_byte4}
```

Figure 4-17: 64-Bit Avalon-ST tx_st_data Cycle Definition for TLP 4-Dword Header with Non-Qword Aligned Address

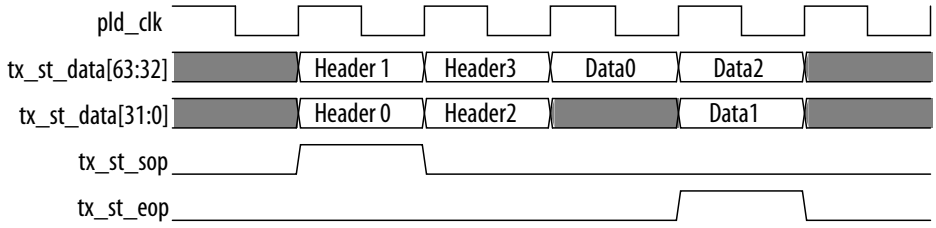


Figure 4-18: 64-Bit Transaction Layer Backpressures the Application Layer

The following figure illustrates the timing of the TX interface when the Cyclone V Hard IP for PCI Express pauses transmission by the Application Layer by deasserting `tx_st_ready`. Because the `readyLatency` is two cycles, the Application Layer deasserts `tx_st_valid` after two cycles and holds `tx_st_data` until two cycles after `tx_st_ready` is asserted.

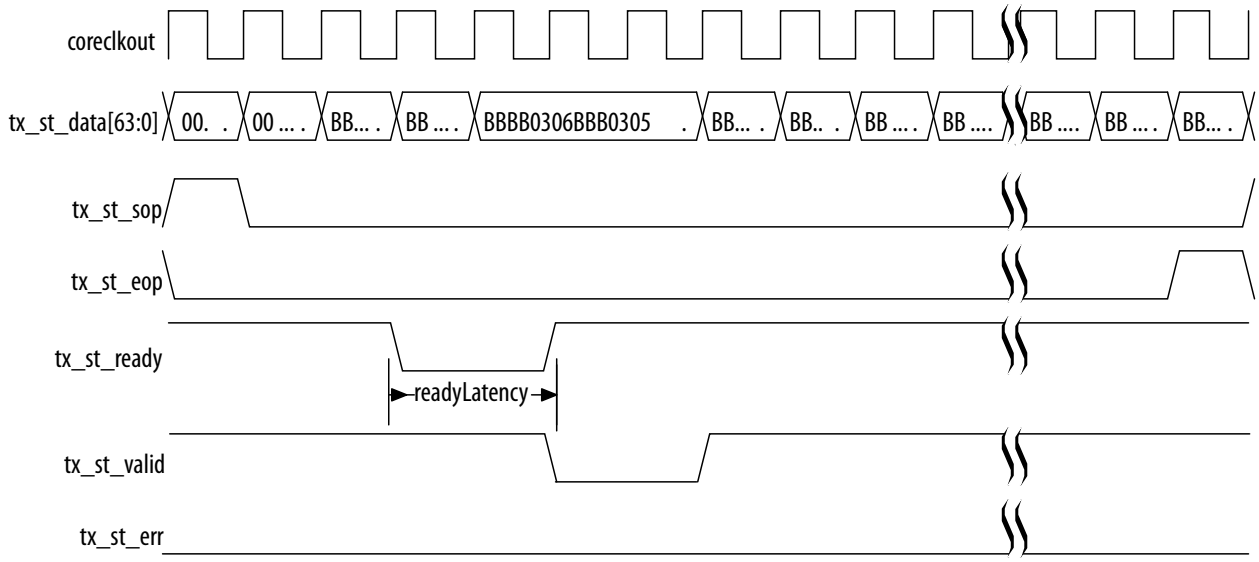
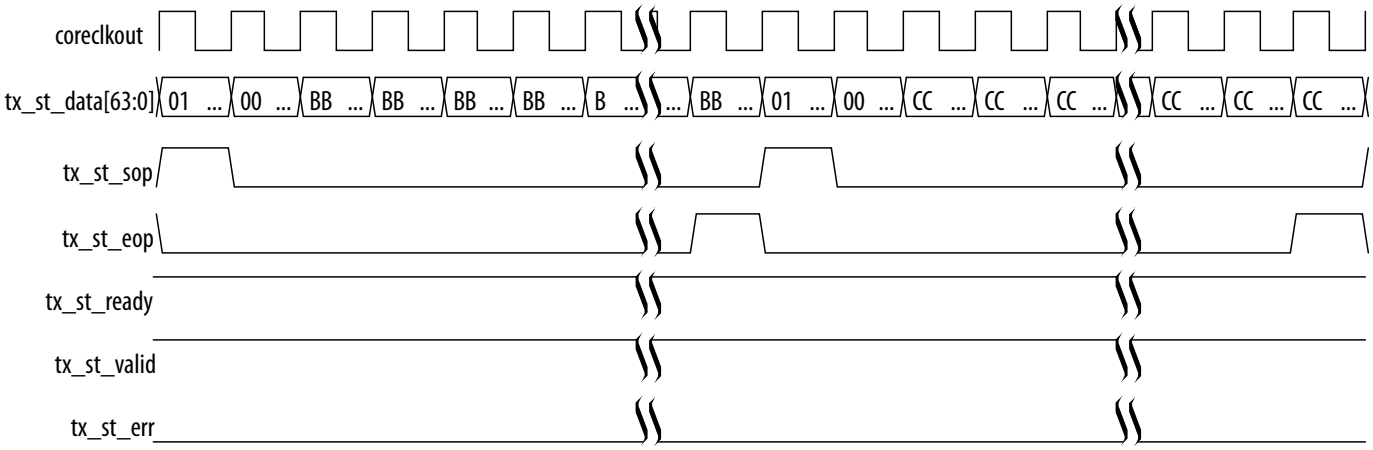


Figure 4-19: 64-Bit Back-to-Back Transmission on the TX Interface

The following figure illustrates back-to-back transmission of 64-bit packets with no idle cycles between the assertion of `tx_st_eop` and `tx_st_sop`.



Data Alignment and Timing for the 128-Bit Avalon-ST TX Interface

Figure 4-20: 128-Bit Avalon-ST tx_st_data Cycle Definition for 3-Dword Header TLP with Qword Aligned Address

The following figure shows the mapping of 128-bit Avalon-ST TX packets to PCI Express TLPs for a three dword header with qword aligned addresses. Assertion of `tx_st_empty` in an `rx_st_eop` cycle indicates valid data in the lower 64 bits of `tx_st_data`.

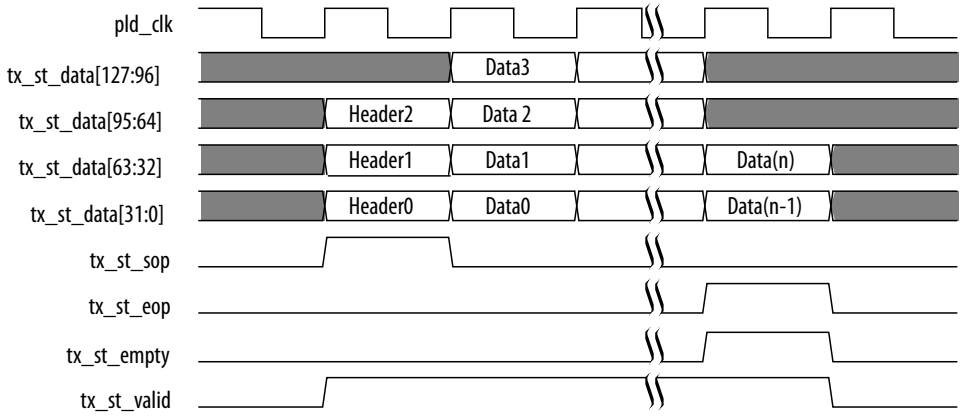


Figure 4-21: 128-Bit Avalon-ST tx_st_data Cycle Definition for 3-Dword Header TLP with non-Qword Aligned Address

The following figure shows the mapping of 128-bit Avalon-ST TX packets to PCI Express TLPs for a 3 dword header with non-qword aligned addresses. It also shows `tx_st_err` assertion.

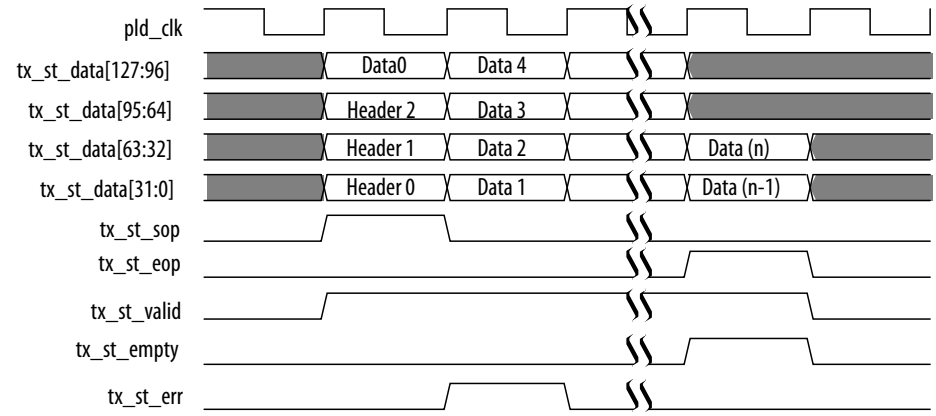


Figure 4-22: 128-Bit Avalon-ST tx_st_data Cycle Definition for 4-Dword Header TLP with Qword Aligned Address

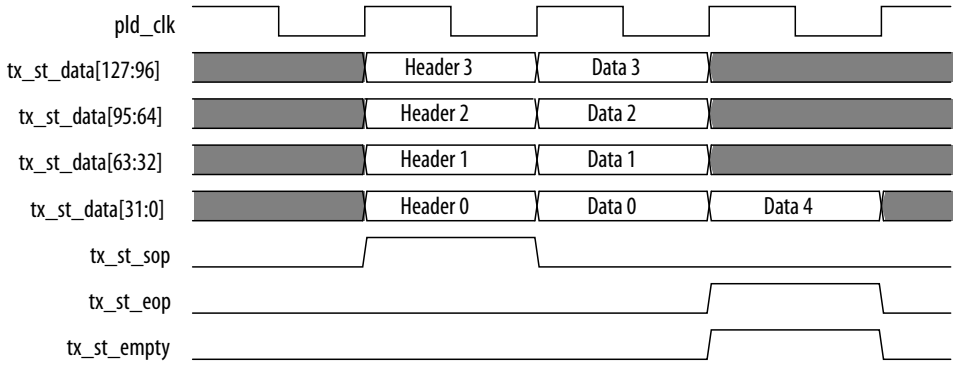


Figure 4-23: 128-Bit Avalon-ST tx_st_data Cycle Definition for 4-Dword Header TLP with non-Qword Aligned Address

The following figure shows the mapping of 128-bit Avalon-ST TX packets to PCI Express TLPs for a four dword header TLP with non-qword aligned addresses. In this example, `tx_st_empty` is low because the data ends in the upper 64 bits of `tx_st_data`.

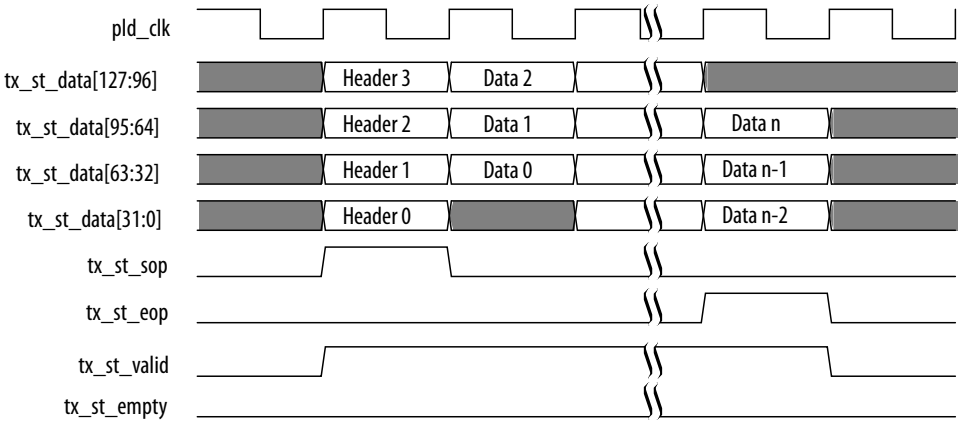


Figure 4-24: 128-Bit Back-to-Back Transmission on the Avalon-ST TX Interface

The following figure illustrates back-to-back transmission of 128-bit packets with idle dead cycles between the assertion of `tx_st_eop` and `tx_st_sop`.

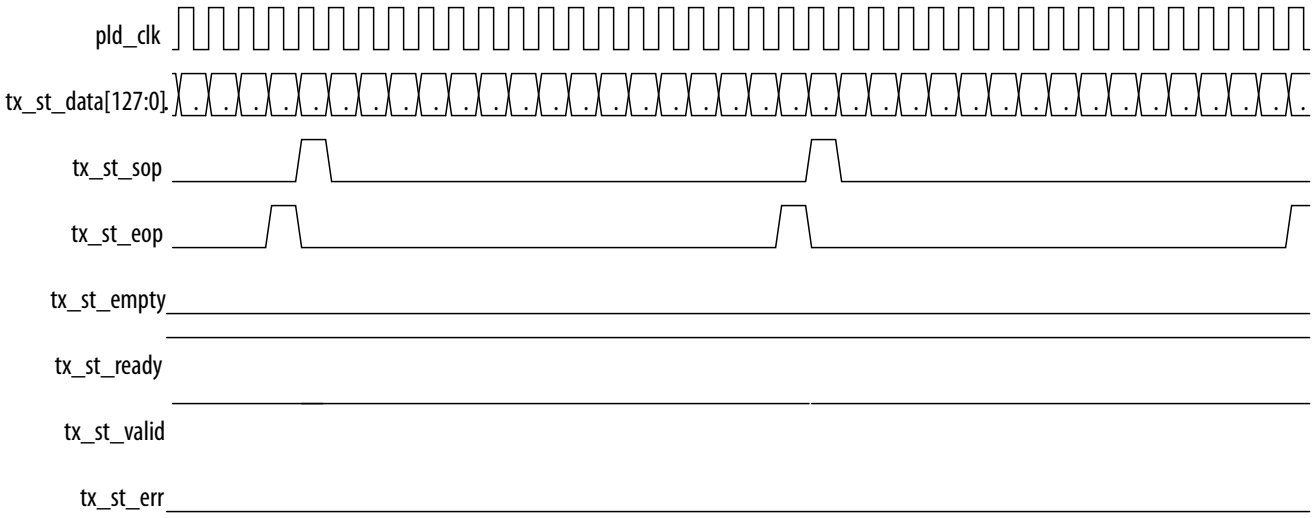
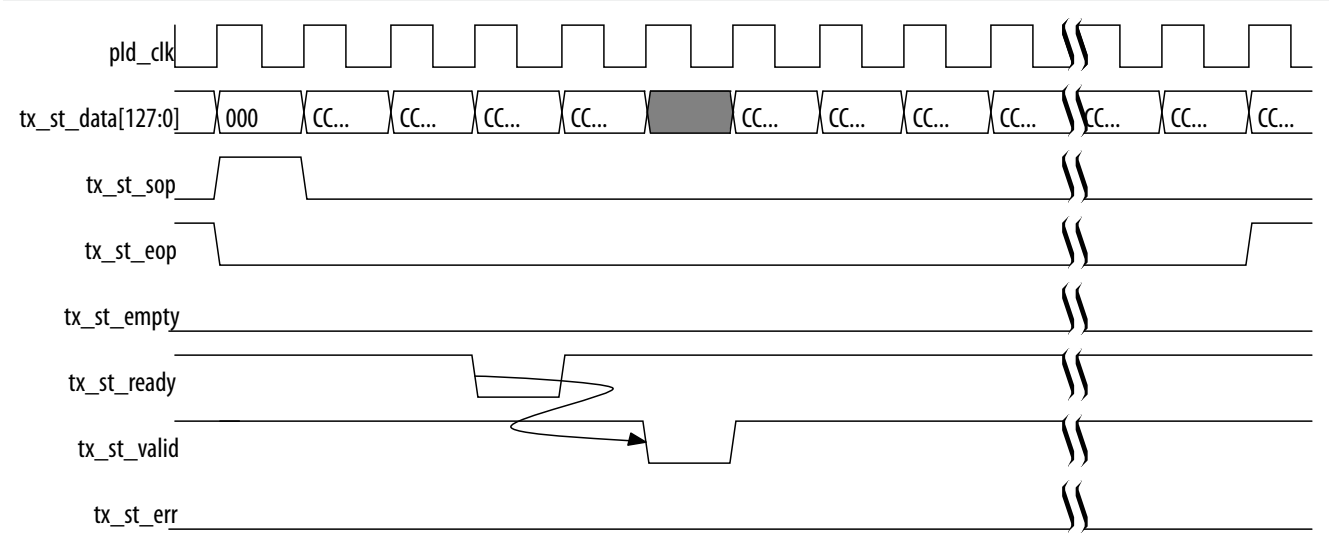


Figure 4-25: 128-Bit Hard IP Backpressures the Application Layer for TX Transactions

The following figure illustrates the timing of the TX interface when the Cyclone V Hard IP for PCI Express pauses the Application Layer by deasserting `tx_st_ready`. Because the `readyLatency` is two cycles, the Application Layer deasserts `tx_st_valid` after two cycles and holds `tx_st_data` until two cycles after `tx_st_ready` is reasserted



Root Port Mode Configuration Requests

If your Application Layer implements ECRC forwarding, it should not apply ECRC forwarding to Configuration Type 0 packets that it issues on the Avalon-ST interface. There should be no ECRC appended to the TLP, and the TD bit in the TLP header should be set to 0. These packets are processed internally by the Hard IP block and are not transmitted on the PCI Express link.

To ensure proper operation when sending Configuration Type 0 transactions in Root Port mode, the application should wait for the Configuration Type 0 transaction to be transferred to the Hard IP for PCI Express Configuration Space before issuing another packet on the Avalon-ST TX port. You can do this by waiting for the core to respond with a completion on the Avalon-ST RX port before issuing the next Configuration Type 0 transaction.

Clock Signals

Table 4-5: Clock Signals

Signal	Direction	Description
refclk	Input	Reference clock for the IP core. It must have the frequency specified under the **System Settings** heading in the parameter editor. This is a dedicated free running input clock to the dedicated REFCLK pin. If your design meets the following criteria: • Enables CvP • Includes an additional transceiver PHY connected to the same Transceiver Reconfiguration Controller then you must connect refclk to the mgmt_clk_clk signal of the Transceiver Reconfiguration Controller and the additional transceiver PHY. In addition, if your design includes more than one Transceiver Reconfiguration Controller on the same side of the FPGA, they all must share the mgmt_clk_clk signal.
pld_clk	Input	Clocks the Application Layer. You can drive this clock with coreclkout_hip. If you drive pld_clk with another clock source, it must be equal to or faster than coreclkout_hip.
coreclkout	Output	This is a fixed frequency clock used by the Data Link and Transaction Layers. To meet PCI Express link bandwidth constraints, this clock has minimum frequency requirements as listed in *Application Layer Clock Frequency for All Combination of Link Width, Data Rate and Application Layer Interface Width* in the *Reset and Clocks* chapter .

Related Information

Clocks on page 6-5

Reset, Status, and Link Training Signals

Refer to *Reset and Clocks* for more information about the reset sequence and a block diagram of the reset logic.

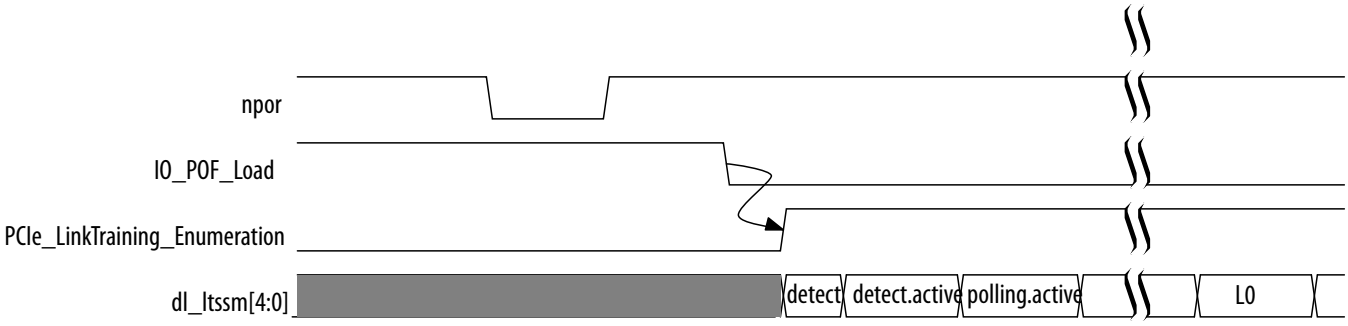
Table 4-6: Reset Signals

Signal	Direction	Description
npor	Input	Active low reset signal. In the Altera hardware example designs, npor is the OR of pin_perst and local_rstn coming from the software Application Layer. If you do not drive a soft reset signal from the Application Layer, this signal must be derived from pin_perst. You cannot disable this signal. Resets the entire IP Core and transceiver. Asynchronous. In systems that use the hard reset controller, this signal is *edge, not level* sensitive; consequently, you cannot use a low value on this signal to hold custom logic in reset. For more information about the hard and soft reset controllers, refer to *Reset*.
reset_status	Output	Active high reset status signal. When asserted, this signal indicates that the Hard IP clock is in reset. The reset_status signal is synchronous to the pld_clk clock and is deasserted only when the npor is deasserted and the Hard IP for PCI Express is not in reset (reset_status_hip = 0). You should use reset_status to drive the reset of your application. This reset is used for the Hard IP for PCI Express IP Core with the Avalon-ST interface.
pin_perst	Input	Active low reset from the PCIe reset pin of the device. pin_perst resets the datapath and control registers. Configuration via Protocol (CvP) requires this signal. For more information about CvP refer to *Configuration via Protocol (CvP)*. Cyclone V devices can have up to 4 instances of the Hard IP for PCI Express. Each instance has its own pin_perst signal. *You must connect the* pin_perst *of each Hard IP instance to the corresponding* nPERST *pin of the device.* These pins have the following locations: • NPERSTL0: bottom left Hard IP and CvP blocks • NPERSTL1: top left Hard IP block • NPERSTR0: bottom right Hard IP block • NPERSTR1: top right Hard IP block For example, if you are using the Hard IP instance in the bottom left corner of the device, you must connect pin_perst to NPERSL0. For maximum use of the Cyclone V device, Altera recommends that you use the bottom left Hard IP first. This is the only location that supports CvP over a PCIe link. If your design does not require CvP, you may select other Hard IP blocks.

Signal	Direction	Description
		Refer to the appropriate device pinout for correct pin assignment for more detailed information about these pins. The *PCI Express Card Electromechanical Specification 2.0* specifies this pin requires 3.3 V. You can drive this 3.3V signal to the nPERST* even if the V_{CCPGM} of the bank is not 3.3V if the following 2 conditions are met: • The input signal meets the V_{IH} and V_{IL} specification for LVTTL.

Figure 4-26: Reset and Link Training Timing Relationships

The following figure illustrates the timing relationship between npor and the LTSSM L0 state.



Note: To meet the 100 ms system configuration time, you must use the fast passive parallel configuration scheme with CvP and a 32-bit data width (FPP x32) or use the CvP in autonomous mode.

Table 4-7: Status and Link Training Signals

Signal	Direction	Description
serdes_pll_locked	Output	When asserted, indicates that the PLL that generates the coreclkout_hip clock signal is locked. In pipe simulation mode this signal is always asserted.
pld_core_ready	Input	When asserted, indicates that the Application Layer is ready for operation and is providing a stable clock to the pld_clk input. If the coreclkout_hip Hard IP output clock is sourcing the pld_clk Hard IP input, this input can be connected to the serdes_pll_locked output.
pld_clk_inuse	Output	When asserted, indicates that the Hard IP Transaction Layer is using the pld_clk as its clock and is ready for operation with the Application Layer. For reliable operation, hold the Application Layer in reset until pld_clk_inuse is asserted.

Signal	Direction	Description
dlup	Output	When asserted, indicates that the Hard IP block is in the Data Link Control and Management State Machine (DLCMSM) DL_Up state.
dlup_exit	Output	This signal is asserted low for one pld_clk cycle when the IP core exits the DLCMSM DL_Up state, indicating that the Data Link Layer has lost communication with the other end of the PCIe link and left the Up state. When this pulse is asserted, the Application Layer should generate an internal reset signal that is asserted for at least 32 cycles.
ev128ns	Output	Asserted every 128 ns to create a time base aligned activity.
ev1us	Output	Asserted every 1µs to create a time base aligned activity.
hotrst_exit	Output	Hot reset exit. This signal is asserted for 1 clock cycle when the LTSSM exits the hot reset state. This signal should cause the Application Layer to be reset. This signal is active low. When this pulse is asserted, the Application Layer should generate an internal reset signal that is asserted for at least 32 cycles.
l2_exit	Output	L2 exit. This signal is active low and otherwise remains high. It is asserted for one cycle (changing value from 1 to 0 and back to 1) after the LTSSM transitions from l2.idle to detect. When this pulse is asserted, the Application Layer should generate an internal reset signal that is asserted for at least 32 cycles.
lane_act[3:0]	Output	Lane Active Mode: This signal indicates the number of lanes that configured during link training. The following encodings are defined: • 4'b0001: 1 lane • 4'b0010: 2 lanes • 4'b0100: 4 lanes • 4'b1000: 8 lanes
currentspeed[1:0]	Output	Indicates the current speed of the PCIe link. The following encodings are defined: • 2b'00: Undefined • 2b'01: Gen1 • 2b'10: Gen2 • 2b'11: Gen3

Signal	Direction	Description
ltssmstate[4:0]	Output	LTSSM state: The LTSSM state machine encoding defines the following states: • 00000: Detect.Quiet • 00001: Detect.Active • 00010: Polling.Active • 00011: Polling.Compliance • 00100: Polling.Configuration • 00101: Polling.Speed • 00110: config.Linkwidthstart • 00111: Config.Linkaccept • 01000: Config.Lanenumaccept • 01001: Config.Lanenumwait • 01010: Config.Complete • 01011: Config.Idle • 01100: Recovery.Rcvlock • 01101: Recovery.Rcvconfig • 01110: Recovery.Idle • 01111: L0 • 10000: Disable • 10001: Loopback.Entry • 10010: Loopback.Active • 10011: Loopback.Exit • 10100: Hot.Reset • 10101: LOs • 11001: L2.transmit.Wake • 11010: Speed.Recovery • 11011: Recovery.Equalization, Phase 0 • 11100: Recovery.Equalization, Phase 1 • 11101: Recovery.Equalization, Phase 2 • 11110: recovery.Equalization, Phase 3

Related Information

- **PCI Express Card Electromechanical Specification 2.0**
- **Configuration via Protocol (CvP) Implementation in Altera FPGAs User Guide**

ECRC Forwarding

On the Avalon-ST interface, the ECRC field follows the same alignment rules as payload data. For packets with payload, the ECRC is appended to the data as an extra dword of payload. For packets without payload, the ECRC field follows the address alignment as if it were a one dword payload. The position of

the ECRC data for data depends on the address alignment. For packets with no payload data, the ECRC position corresponds to the position of `Data0`.

Error Signals

The following table describes the ECC error signals. These signals are all valid for one clock cycle. They are synchronous to `coreclkout_hip`.

ECC for the RX and retry buffers is implemented with MRAM. These error signals are flags. If a specific location of MRAM has errors, as long as that data is in the ECC decoder, the flag indicates the error.

When a correctable ECC error occurs, the Cyclone V Hard IP for PCI Express recovers without any loss of information. No Application Layer intervention is required. In the case of uncorrectable ECC error, Altera recommends that you reset the core.

The Avalon-ST `rx_st_err` indicates an uncorrectable error in the RX buffer. This signal is described in *64- or 128-Bit Avalon-ST RX Datapath* in the *Avalon-ST RX Interface* description.

Table 4-8: Error Signals

Signal	I/O	Description
`derr_cor_ext_rcv0`	Output	Indicates a corrected error in the RX buffer. This signal is for debug only. It is not valid until the RX buffer is filled with data. This is a pulse, not a level, signal. Internally, the pulse is generated with the 500 MHz clock. A pulse extender extends the signal so that the FPGA fabric running at 250 MHz can capture it. Because the error was corrected by the IP core, no Application Layer intervention is required. (1)
`derr_rpl`	Output	Indicates an uncorrectable error in the retry buffer. This signal is for debug only. (1)
`derr_cor_ext_rpl0`	Output	Indicates a corrected ECC error in the retry buffer. This signal is for debug only. Because the error was corrected by the IP core, no Application Layer intervention is required. (1)

Notes:

1. Debug signals are not rigorously verified and should only be used to observe behavior. Debug signals should not be used to drive logic custom logic.

Related Information

Avalon-ST RX Interface

Interrupts for Endpoints

Refer to *Interrupts* for detailed information about all interrupt mechanisms.

Table 4-9: Interrupt Signals for Endpoints

Signal	Direction	Description
app_msi_req	Input	Application Layer MSI request. Assertion causes an MSI posted write TLP to be generated based on the MSI configuration register values and the app_msi_tc and app_msi_num input ports.
app_msi_ack	Output	Application Layer MSI acknowledge. This signal acknowledges the Application Layer's request for an MSI interrupt.
app_msi_tc[2:0]	Input	Application Layer MSI traffic class. This signal indicates the traffic class used to send the MSI (unlike INTX interrupts, any traffic class can be used to send MSIs).
app_msi_num[4:0]	Input	MSI number of the Application Layer. This signal provides the low order message data bits to be sent in the message data field of MSI messages requested by app_msi_req. Only bits that are enabled by the MSI Message Control register apply.
app_int_sts	Input	Controls legacy interrupts. Assertion of app_int_sts causes an Assert_INTA message TLP to be generated and sent upstream. Deassertion of app_int_sts causes a Deassert_INTA message TLP to be generated and sent upstream.

Interrupts for Root Ports

Table 4-10: Interrupt Signals for Root Ports

Signal	Direction	Description
int_status[3:0]	Output	These signals drive legacy interrupts to the Application Layer as follows: • int_status[0]: interrupt signal A • int_status[1]: interrupt signal B • int_status[2]: interrupt signal C • int_status[3]: interrupt signal D
aer_msi_num[4:0]	Input	Advanced error reporting (AER) MSI number. Provides the low-order message data bits to be sent in the message data field of the MSI messages associated with the AER capability structure. Only bits that are enabled by the MSI Message Control register are used. For Root Ports only.

Signal	Direction	Description
pex_msi_num[4:0]	Input	Power management MSI number. This signal provides the low-order message data bits to be sent in the message data field of MSI messages associated with the PCI Express capability structure. Only bits that are enabled by the MSI Message Control register are used. For Root Ports only.
serr_out	Output	System Error: This signal only applies to Root Port designs that report each system error detected, assuming the proper enabling bits are asserted in the Root Control and Device Control registers. If enabled, serr_out is asserted for a single clock cycle when a system error occurs. System errors are described in the *PCI Express Base Specification 2.1 or 3.0* in the Root Control register.

Related Information

PCI Express Base Specification 2.1 or 3.0

Completion Side Band Signals

The following table describes the signals that comprise the completion side band signals for the Avalon-ST interface. The Cyclone V Hard IP for PCI Express provides a completion error interface that the Application Layer can use to report errors, such as programming model errors. When the Application Layer detects an error, it can assert the appropriate cpl_err bit to indicate what kind of error to log. If separate requests result in two errors, both are logged. The Hard IP sets the appropriate status bits for the errors in the Configuration Space, and automatically sends error messages in accordance with the *PCI Express Base Specification*. Note that the Application Layer is responsible for sending the completion with the appropriate completion status value for non-posted requests. Refer to *Error Handling* for information on errors that are automatically detected and handled by the Hard IP.

For a description of the completion rules, the completion header format, and completion status field values, refer to Section 2.2.9 of the *PCI Express Base Specification*.

Table 4-11: Completion Signals for the Avalon-ST Interface

Signal	Direction	Description
cpl_err[6:0]	Input	Completion error. This signal reports completion errors to the Configuration Space. When an error occurs, the appropriate signal is asserted for one cycle.

Signal	Direction	Description
		• `cpl_err[0]`: Completion timeout error with recovery. This signal should be asserted when a master-like interface has performed a non-posted request that never receives a corresponding completion transaction after the 50 ms timeout period when the error is correctable. The Hard IP automatically generates an advisory error message that is sent to the Root Complex. • `cpl_err[1]`: Completion timeout error without recovery. This signal should be asserted when a master-like interface has performed a non-posted request that never receives a corresponding completion transaction after the 50 ms time-out period when the error is not correctable. The Hard IP automatically generates a non-advisory error message that is sent to the Root Complex. • `cpl_err[2]`: Completer abort error. The Application Layer asserts this signal to respond to a non-posted request with a Completer Abort (CA) completion. The Application Layer generates and sends a completion packet with Completer Abort (CA) status to the requestor and then asserts this error signal to the Hard IP. The Hard IP automatically sets the error status bits in the Configuration Space register and sends error messages in accordance with the *PCI Express Base Specification*. • `cpl_err[3]`: Unexpected completion error. This signal must be asserted when an Application Layer master block detects an unexpected completion transaction. Many cases of unexpected completions are detected and reported internally by the Transaction Layer. For a list of these cases, refer to *Transaction Layer Errors*.

Signal	Direction	Description

		• cpl_err[4]: Unsupported Request (UR) error for posted TLP. The Application Layer asserts this signal to treat a posted request as an Unsupported Request. The Hard IP automatically sets the error status bits in the Configuration Space register and sends error messages in accordance with the *PCI Express Base Specification*. Many cases of Unsupported Requests are detected and reported internally by the Transaction Layer. For a list of these cases, refer to *Transaction Layer Errors*.
		• cpl_err[5]: Unsupported Request error for non-posted TLP. The Application Layer asserts this signal to respond to a non-posted request with an Request (UR) completion. In this case, the Application Layer sends a completion packet with the Unsupported Request status back to the requestor, and asserts this error signal. The Hard IP automatically sets the error status bits in the Configuration Space Register and sends error messages in accordance with the *PCI Express Base Specification*. Many cases of Unsupported Requests are detected and reported internally by the Transaction Layer. For a list of these cases, refer to *Transaction Layer Errors*.
		• cpl_err[6]: Log header. If header logging is required, this bit must be set in the every cycle in which any of cpl_err[2], cpl_err[3], cpl_err[4], or cpl_err[5]is set. The Application Layer presents the header to the Hard IP by writing the following values to the following 4 registers using LMI before asserting cpl_err[6]:. The Application Layer presents the header to the Hard IP by writing the following values to the following 4 registers using LMI before asserting cpl_err[6]:
		  • lmi_addr: 12'h81C, lmi_din: err_desc_func0[127:96]
		  • lmi_addr: 12'h820, lmi_din: err_desc_func0[95:64]
		  • lmi_addr: 12'h824, lmi_din: err_desc_func0[63:32]
		  • lmi_addr: 12'h828, lmi_din: err_desc_func0[31:0]

cpl_pending[7:0]	Input	Completion pending. The Application Layer must assert this signal when a master block is waiting for completion, for example, when a transaction is pending. This is a level sensitive input. A bit is provided for each function, where bit 0 corresponds to function 0, and so on.
cpl_err_func[2:0]		Specifies which function is requesting the cpl_err. Must be asserted when cpl_err asserts. Due to clock-domain synchronization circuitry, cpl_err is limited to at most 1 assertion every 8 pld_clk cycles. Whenever cpl_err is asserted, cpl_err_func[2:0] should be updated in the same cycle.

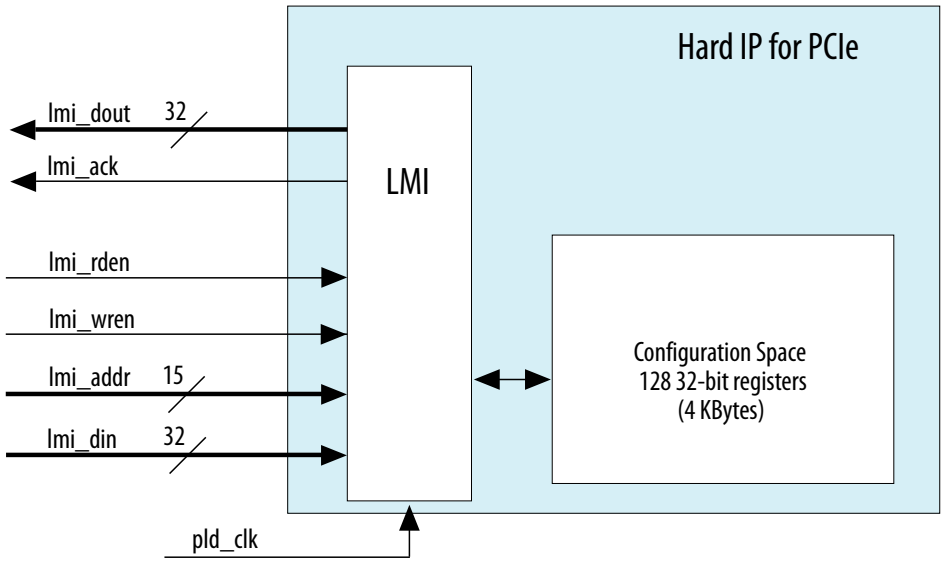
Related Information

Transaction Layer Errors on page 8-3

LMI Signals

LMI interface is used to write log error descriptor information in the TLP header log registers. The LMI access to other registers is intended for debugging, not normal operation.

Figure 4-27: Local Management Interface



The LMI interface is synchronized to `pld_clk` and runs at frequencies up to 250 MHz. The LMI address is the same as the Configuration Space address. The read and write data are always 32 bits. The LMI interface provides the same access to Configuration Space registers as Configuration TLP requests. Register bits have the same attributes, (read only, read/write, and so on) for accesses from the LMI interface and from Configuration TLP requests.

Note: You can also use the Configuration Space signals to read Configuration Space registers. For more information, refer to *Transaction Layer Configuration Space Signals*.

When a LMI write has a timing conflict with configuration TLP access, the configuration TLP accesses have higher priority. LMI writes are held and executed when configuration TLP accesses are no longer pending. An acknowledge signal is sent back to the Application Layer when the execution is complete.

All LMI reads are also held and executed when no configuration TLP requests are pending. The LMI interface supports two operations: local read and local write. The timing for these operations complies with the Avalon-MM protocol described in the *Avalon Interface Specifications*. LMI reads can be issued at any time to obtain the contents of any Configuration Space register. LMI write operations are not recommended for use during normal operation. The Configuration Space registers are written by requests received from the PCI Express link and there may be unintended consequences of conflicting updates from the link and the LMI interface. LMI Write operations are provided for AER header logging, and debugging purposes only.

- In Root Port mode, do not access the Configuration Space using TLPs and the LMI bus simultaneously.

Table 4-12: LMI Interface

Signal	Direction	Description
lmi_dout[31:0]	Output	Data outputs.
lmi_rden	Input	Read enable input.
lmi_wren	Input	Write enable input.
lmi_ack	Output	Write execution done/read data valid.
lmi_addr[11:0]	Input	Address inputs, [1:0] not used.
lmi_din[31:0]	Input	Data inputs.

Figure 4-28: LMI Read

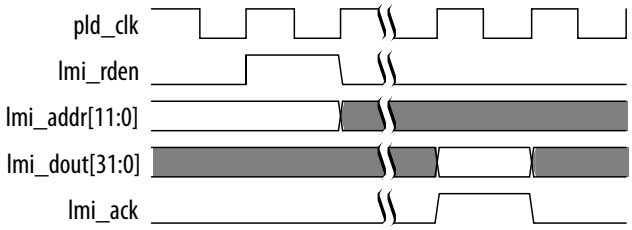
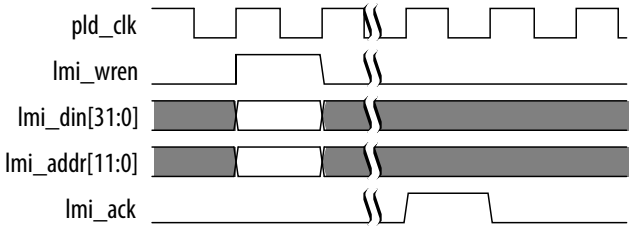


Figure 4-29: LMI Write

Only writeable configuration bits are overwritten by this operation. Read-only bits are not affected. LMI write operations are not recommended for use during normal operation with the exception of AER header logging.



Related Information

Avalon Interface Specifications

Transaction Layer Configuration Space Signals

Table 4-13: Configuration Space Signals

These signals are not available if Configuration Space Bypass mode is enabled.

Signal	Direction	Description
tl_cfg_add[3:0]	0utput	Address of the register that has been updated. This signal is an index indicating which Configuration Space register information is being driven onto tl_cfg_ctl. The indexing is defined in *Multiplexed Configuration Register Information Available on tl_cfg_ctl*. The index increments on every pld_clk cycle.
tl_cfg_ctl[31:0]	0utput	The tl_cfg_ctl signal is multiplexed and contains the contents of the Configuration Space registers. The indexing is defined in *Multiplexed Configuration Register Information Available on tl_cfg_ctl*.
tl_cfg_sts[52:0]	0utput	Configuration status bits. This information updates every pld_clk cycle. The following table provides detailed descriptions of the status bits.
hpg_ctrler[4:0]	Input	The hpg_ctrler signals are only available in Root Port mode and when the Slot capability register is enabled. Refer to the Slot register and Slot capability register parameters in Table 6–9 on page 6–10. For Endpoint variations the hpg_ctrler input should be hardwired to 0s. The bits have the following meanings:
	Input	• [0]: Attention button pressed. This signal should be asserted when the attention button is pressed. If no attention button exists for the slot, this bit should be hardwired to 0, and the Attention Button Present bit (bit[0]) in the Slot capability register parameter is set to 0.
	Input	• [1]: Presence detect. This signal should be asserted when a presence detect circuit detects a presence detect change in the slot.
	Input	• [2]: Manually-operated retention latch (MRL) sensor changed. This signal should be asserted when an MRL sensor indicates that the MRL is Open. If an MRL Sensor does not exist for the slot, this bit should be hardwired to 0, and the MRL Sensor Present bit (bit[2]) in the Slot capability register parameter is set to 0.

Signal	Direction	Description
	Input	• [3]: Power fault detected. This signal should be asserted when the power controller detects a power fault for this slot. If this slot has no power controller, this bit should be hardwired to 0, and the `Power Controller Present` bit (bit[1]) in the Slot capability register parameter is set to 0.
	Input	• [4]: Power controller status. This signal is used to set the command completed bit of the `Slot Status` register. Power controller status is equal to the power controller control signal. If this slot has no power controller, this bit should be hardwired to 0 and the `Power Controller Present` bit (bit[1]) in the Slot capability register is set to 0.

Table 4-14: Mapping Between tl_cfg_sts and Configuration Space Registers

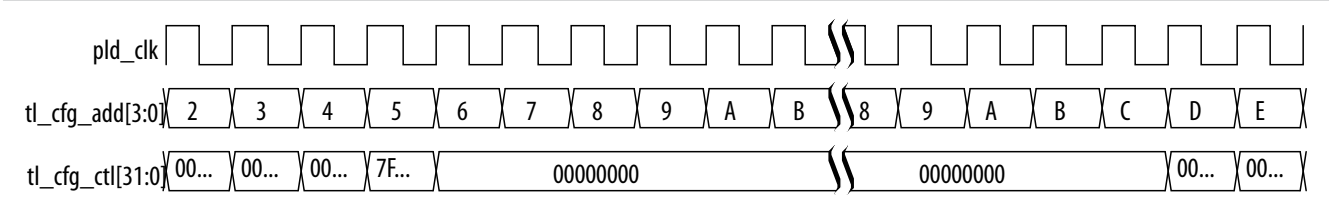
tl_cfg_sts	Configuration Space Register	Description
[52:49]	Device Status Register[3:0]	Records the following errors: • Bit 3: unsupported request detected • Bit 2: fatal error detected • Bit 1: non-fatal error detected • Bit 0: correctable error detected
[48]	Slot Status Register[8]	Data Link Layer state changed
[47]	Slot Status Register[4]	Command completed. (The hot plug controller completed a command.)
[46:31]	Link Status Register[15:0]	Records the following link status information: • Bit 15: link autonomous bandwidth status • Bit 14: link bandwidth management status • Bit 13: Data Link Layer link active • Bit 12: Slot clock configuration • Bit 11: Link Training • Bit 10: Undefined • Bits[9:4]: Negotiated Link Width • Bits[3:0] Link Speed
[30]	Link Status 2 Register[0]	Current de-emphasis level.

tl_cfg_sts	Configuration Space Register	Description
[29:25]	Status Register[15:11]	Records the following 5 primary command status errors: • Bit 15: detected parity error • Bit 14: signaled system error • Bit 13: received master abort • Bit 12: received target abort • Bit 11: signalled target abort
[24]	Secondary Status Register[8]	Master data parity error
[23:6]	Root Status Register[17:0]	Records the following PME status information: • Bit 17: PME pending • Bit 16: PME status • Bits[15:0]: PME request ID[15:0]
[5:1]	Secondary Status Register[15:11]	Records the following 5 secondary command status errors: • Bit 15: detected parity error • Bit 14: received system error • Bit 13: received master abort • Bit 12: received target abort • Bit 11: signalled target abort
[0]	Secondary Status Register[8]	Master Data Parity Error

Configuration Space Register Access Timing

Figure 4-30: tl_cfg_ctl Timing

The following figure shows typical traffic on the `tl_cfg_ctl` bus. The `tl_cfg_add` index increments on the rising edge of the `pld_clk`. The address specifies which Configuration Space register data value is being driven onto `tl_cfg_ctl`.



Configuration Space Register Access

The `tl_cfg_ctl` signal is a multiplexed bus that contains the contents of Configuration Space registers as shown in the figure below. Information stored in the Configuration Space is accessed in round robin order where `tl_cfg_add` indicates which register is being accessed. The following table shows the layout of configuration information that is multiplexed on `tl_cfg_ctl`.

Figure 4-31: Multiplexed Configuration Register Information Available on tl_cfg_ctl

Fields in blue are available only for Root Ports.

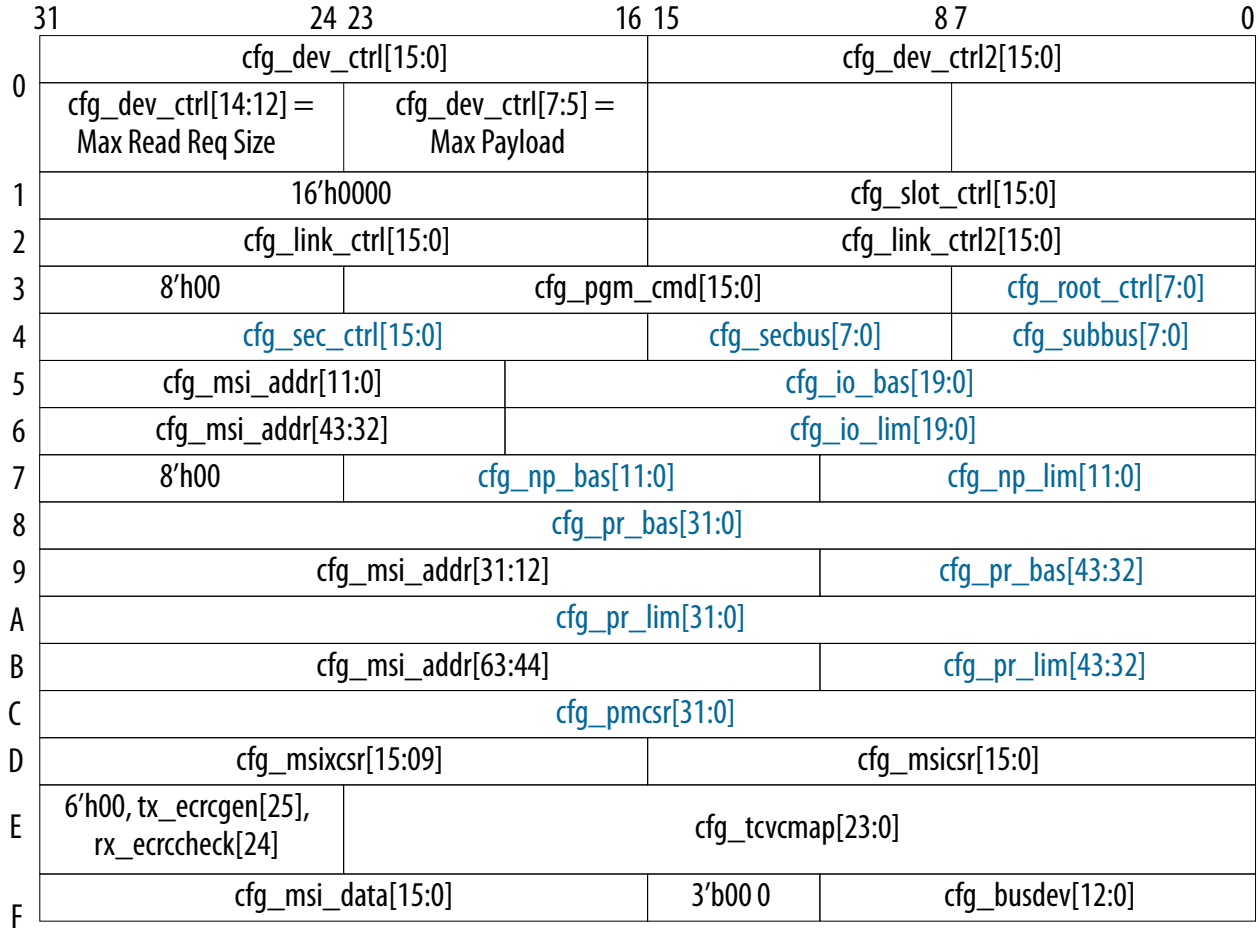


Table 4-15: Configuration Space Register Descriptions

Register	Width	Direction	Description
`cfg_dev_ctrl_func<n>`	16	Output	`cfg_dev_ctrl_func<n>[15:0]` is Device Control register for the PCI Express capability structure.
`cfg_dev_ctrl2`	16	Output	`cfg_dev2ctrl[15:0]` is Device Control 2 for the PCI Express capability structure.

Register	Width	Direction	Description
cfg_slot_ctrl	16	Output	cfg_slot_ctrl[15:0] is the Slot Status of the PCI Express capability structure. This register is only available in Root Port mode.
cfg_link_ctrl	16	Output	cfg_link_ctrl[15:0]is the primary Link Control of the PCI Express capability structure. For Gen2 operation, you must write a 1'b1 to the Retrain Link bit (Bit[5] of the cfg_link_ctrl) of the Root Port to initiate retraining to a higher data rate after the initial link training to Gen1 L0 state. Retraining directs the LTSSM to the Recovery state. Retraining to a higher data rate is not automatic for the Cyclone V Hard IP for PCI Express IP Core even if both devices on the link are capable of a higher data rate.
cfg_link_ctrl2	16	Output	cfg_link_ctrl2[31:16] is the secondary Link Control register of the PCI Express capability structure for Gen2 operation. When tl_cfg_addr=4'b0010, tl_cfg_ctl returns the primary and secondary Link Control registers, {{cfg_link_ctrl[15:0], cfg_link_ctrl2[15:0]}. The primary Link Status register contents are available on tl_cfg_sts[46:31]. For Gen1 variants, the link bandwidth notification bit is always set to 0. For Gen2 variants, this bit is set to 1.
cfg_prm_cmd_func<*n*>	16	Output	Base/Primary Command register for the PCI Configuration Space.
cfg_root_ctrl	8	Output	Root control and status register of the PCI Express capability. This register is only available in Root Port mode.
cfg_sec_ctrl	16	Output	Secondary bus Control and Status register of the PCI Express capability. This register is available only in Root Port mode.
cfg_secbus	8	Output	Secondary bus number. This register is available only in Root Port mode.
cfg_subbus	8	Output	Subordinate bus number. This register is available only in Root Port mode.

Register	Width	Direction	Description
cfg_msi_addr	64	Output	cfg_msi_add[63:32] is the message signaled interrupt (MSI) upper message address. cfg_msi_add[31:0] is the MSI message address.
cfg_io_bas	20	Output	The upper 20 bits of the I/O limit registers of the Type1 Configuration Space. This register is only available in Root Port mode.
cfg_io_lim	20	Output	The upper 20 bits of the IO limit registers of the Type1 Configuration Space. This register is only available in Root Port mode.
cfg_np_bas	12	Output	The upper 12 bits of the memory base register of the Type1 Configuration Space. This register is only available in Root Port mode.
cfg_np_lim	12	Output	The upper 12 bits of the memory limit register of the Type1 Configuration Space. This register is only available in Root Port mode.
cfg_pr_bas	44	Output	The upper 44 bits of the prefetchable base registers of the Type1 Configuration Space. This register is only available in Root Port mode.
cfg_pr_lim	44	Output	The upper 44 bits of the prefetchable limit registers of the Type1 Configuration Space. Available in Root Port mode.
cfg_pmcsr	32	Output	cfg_pmcsr[31:16] is Power Management Control and cfg_pmcsr[15:0]is the Power Management Status register.
cfg_msixcsr	16	Output	MSI-X message control.
cfg_msicsr	16	Output	MSI message control. Refer to the following table for the fields of this register.

Register	Width	Direction	Description
cfg_tcvcmap	24	Output	Configuration traffic class (TC)/virtual channel (VC) mapping. The Application Layer uses this signal to generate a TLP mapped to the appropriate channel based on the traffic class of the packet. • cfg_tcvcmap[2:0]: Mapping for TC0 (always 0) . • cfg_tcvcmap[5:3]: Mapping for TC1. • cfg_tcvcmap[8:6]: Mapping for TC2. • cfg_tcvcmap[11:9]: Mapping for TC3. • cfg_tcvcmap[14:12]: Mapping for TC4. • cfg_tcvcmap[17:15]: Mapping for TC5. • cfg_tcvcmap[20:18]: Mapping for TC6. • cfg_tcvcmap[23:21]: Mapping for TC7.
cfg_msi_data	16	Output	cfg_msi_data[15:0] is message data for MSI.
cfg_busdev	13	Output	Bus/Device Number captured by or programmed in the Hard IP.

Figure 4-32: Configuration MSI Control Status Register

Field and Bit Map					
15 9	8	7	6 4	3 1	0
reserved	mask capability	64-bit address capability	multiple message enable	multiple message capable	MSI enable

Table 4-16: Configuration MSI Control Status Register Field Descriptions

Bit(s)	Field	Description
[15:9]	Reserved	N/A
[8]	mask capability	Per-vector masking capable. This bit is hardwired to 0 because the function does not support the optional MSI per-vector masking using the Mask_Bits and Pending_Bits registers defined in the *PCI Local Bus Specification*. Per-vector masking can be implemented using Application Layer registers.
[7]	64-bit address capability	64-bit address capable. • 1: function capable of sending a 64-bit message address • 0: function not capable of sending a 64-bit message address

Bit(s)	Field	Description
[6:4]	multiple message enable	This field indicates permitted values for MSI signals. For example, if "100" is written to this field 16 MSI signals are allocated. • 3'b000: 1 MSI allocated • 3'b001: 2 MSI allocated • 3'b010: 4 MSI allocated • 3'b011: 8 MSI allocated • 3'b100: 16 MSI allocated • 3'b101: 32 MSI allocated • 3'b110: Reserved • 3'b111: Reserved
[3:1]	multiple message capable	This field is read by system software to determine the number of requested MSI messages. • 3'b000: 1 MSI requested • 3'b001: 2 MSI requested • 3'b010: 4 MSI requested • 3'b011: 8 MSI requested • 3'b100: 16 MSI requested • 3'b101: 32 MSI requested • 3'b110: Reserved
[0]	MSI Enable	If set to 0, this component is not permitted to use MSI.

Related Information

- **PCI Express Base Specification 2.1 or 3.0**
- **PCI Local Bus Specification, Rev. 3.0**

Hard IP Reconfiguration Interface

The Hard IP reconfiguration interface is an Avalon-MM slave interface with a 10-bit address and 16-bit data bus. You can use this bus to dynamically modify the value of configuration registers that are read-only at run time. To ensure proper system operation, reset or repeat device enumeration of the PCI Express link after changing the value of read-only configuration registers of the Hard IP.

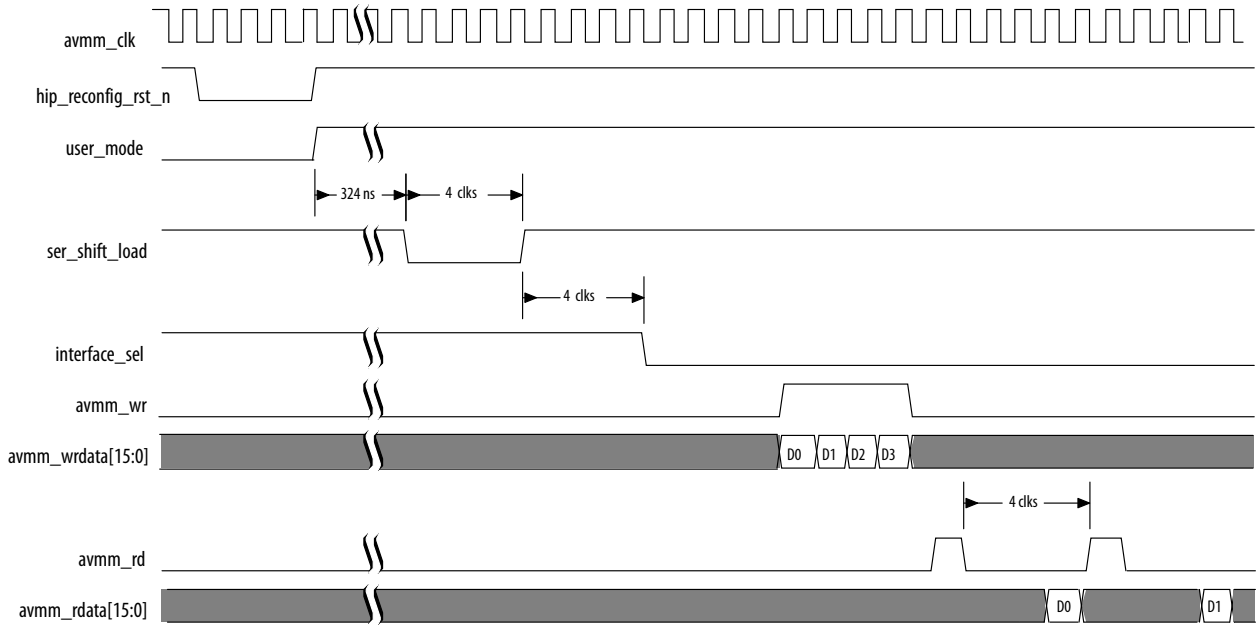
For an example that illustrates how to use this interface, refer to *PCI SIG Gen2 x8 Merged Design - Stratix V* on the Altera wiki. The *Related Information* section below provides a link to this example.

Table 4-17: Hard IP Reconfiguration Signals

Signal	Direction	Description
hip_reconfig_clk	Input	

Signal	Direction	Description
hip_reconfig_rst_n	Input	Active-low Avalon-MM reset. Resets all of the dynamic reconfiguration registers to their default values as described in *Hard IP Reconfiguration Registers*.
hip_reconfig_address[9:0]	Input	The 10-bit reconfiguration address.
hip_reconfig_read	Input	Read signal. This interface is not pipelined. You must wait for the return of the hip_reconfig_readdata[15:0] from the current read before starting another read operation.
hip_reconfig_readdata[15:0]	Output	16-bit read data. hip_reconfig_readdata[15:0] is valid on the third cycle after the assertion of hip_reconfig_read.
hip_reconfig_write	Input	Write signal.
hip_reconfig_writedata[15:0]	Input	16-bit write model.
hip_reconfig_byte_en[1:0]	Input	Byte enables, currently unused.
ser_shift_load	Input	You must toggle this signal once after changing to user mode before the first access to read-only registers. This signal should remain asserted for a minimum of 324 ns after switching to user mode.
interface_sel	Input	A selector which must be asserted when performing dynamic reconfiguration. Drive this signal low 4 clock cycles after the release of ser_shif t_load.

Figure 4-33: Hard IP Reconfiguration Bus Timing of Read-Only Registers



For a detailed description of the Avalon-MM protocol, refer to the *Avalon Memory Mapped Interfaces* chapter in the *Avalon Interface Specifications*.

Related Information

- **Avalon Interface Specifications**
- **PCI SIG Gen2 x8 Merged Design - Stratix V**

Power Management Signals

Table 4-18: Power Management Signals

Signal	Direction	Description
pme_to_cr	Input	Power management turn off control register. Root Port—When this signal is asserted, the Root Port sends the PME_turn_off message. Endpoint—This signal is asserted to acknowledge the PME_turn_off message by sending pme_to_ack to the Root Port.

Signal	Direction	Description
pme_to_sr	Output	Power management turn off status register. Root Port—This signal is asserted for 1 clock cycle when the Root Port receives the pme_turn_off acknowledge message. Endpoint—This signal is asserted for 1 cycle when the Endpoint receives the PME_turn_off message from the Root Port.
pm_event	Input	Power Management Event. This signal is only available for Endpoints. The Endpoint initiates a a power_management_event message (PM_PME) that is sent to the Root Port. If the Hard IP is in a low power state, the link exits from the low-power state to send the message. This signal is positive edge-sensitive.
pm_event_func[2:0]	Input	Specifies the function associated with a Power Management Event.
pm_data[9:0]	Input	Power Management Data. This bus indicates power consumption of the component. This bus can only be implemented if all three bits of AUX_power (part of the Power Management Capabilities structure) are set to 0. This bus includes the following bits: • pm_data[9:2]: Data Register: This register maintains a value associated with the power consumed by the component. (Refer to the example below) • pm_data[1:0]: Data Scale: This register maintains the scale used to find the power consumed by a particular component and can include the following values: • 2b'00: unknown • 2b'01: 0.1 × • 2b'10: 0.01 × • 2b'11: 0.001 × For example, the two registers might have the following values: • pm_data[9:2]: b'1110010 = 114 • pm_data[1:0]: b'10, which encodes a factor of 0.01 To find the maximum power consumed by this component, multiply the data value by the data Scale (114 × .01 = 1.14). 1.14 watts is the maximum power allocated to this component in the power state selected by the data_select field.
pm_auxpwr	Input	Power Management Auxiliary Power: This signal can be tied to 0 because the L2 power state is not supported.

Figure 4-34: Layout of Power Management Capabilities Register

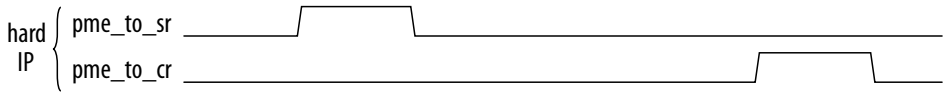
31 24	23 16	15	14 13	12 9	8	7 2	1 0
data register	reserved	PME_status	data_scale	data_select	PME_EN	reserved	PM_state

Table 4-19: Power Management Capabilities Register Field Descriptions

Bits	Field	Description
[31:24]	Data register	This field indicates in which power states a function can assert the PME# message.
[23:16]	reserved	—
[15]	PME_status	When set to 1, indicates that the function would normally assert the PME# message independently of the state of the PME_en bit.
[14:13]	data_scale	This field indicates the scaling factor when interpreting the value retrieved from the data register. This field is read-only.
[12:9]	data_select	This field indicates which data should be reported through the data register and the data_scale field.
[8]	PME_EN	1: indicates that the function can assert PME#0: indicates that the function cannot assert PME#
[7:2]	reserved	—
[1:0]	PM_state	Specifies the power management state of the operating condition being described. The following encodings are defined: • 2b'00 D0 • 2b'01 D1 • 2b'10 D2 • 2b'11 D3 A device returns 2b'11 in this field and Aux or PME Aux in the type register to specify the *D3-Cold PM* state. An encoding of 2b'11 along with any other type register value specifies the *D3-Hot* state.

Figure 4-35: pme_to_sr and pme_to_cr in an Endpoint IP core

The following figure illustrates the behavior of `pme_to_sr` and `pme_to_cr` in an Endpoint. First, the Hard IP receives the `PME_turn_off` message which causes `pme_to_sr` to assert. Then, the Application Layer sends the `PME_to_ack` message to the Root Port by asserting `pme_to_cr`.



Physical Layer Interface Signals

Altera provides an integrated solution with the Transaction, Data Link and Physical Layers. The IP Parameter Editor generates a SERDES variation file, **<*variation*>_serdes.v** or **.vhd** , in addition to the Hard IP variation file, **<*variation*>.v** or **.vhd**. The SERDES entity is included in the library files for PCI Express.

Serial Data Signals

Table 4-20: 1-Bit Interface Signals

Signal	Direction	Description
`tx_out[7:0]` (1)	Output	Transmit output. These signals are the serial outputs of lanes 7–0.
`rx_in[7:0]` (1)	Input	Receive input. These signals are the serial inputs of lanes 7–0.

Note:

1. The x1 IP core only has lane 0. The x2 IP core only has lanes 1–0. The x4 IP core only has lanes 3–0.

Refer to *Pin-out Files for Altera Devices* for pin-out tables for all Altera devices in **.pdf**, **.txt**, and **.xls** formats.

Transceiver channels are arranged in groups of six. For GX devices, the lowest six channels on the left side of the device are labeled GXB_L0, the next group is GXB_L1, and so on. Channels on the right side of the device are labeled GXB_R0, GXB_R1, and so on. Be sure to connect the Hard IP for PCI Express on the left side of the device to appropriate channels on the left side of the device, as specified in the *Pin-out Files for Altera Devices*.

Related Information

Pin-out Files for Altera Devices

Physical Layout of Hard IP in Cyclone V Devices

Cyclone V devices include one or two Hard IP for PCI Express IP cores. The following figures illustrate the placement of the PCIe IP cores, transceiver banks, and channels. Note that the bottom left IP core includes the CvP functionality. The other Hard IP blocks do not include the CvP functionality. Transceiver banks include six channels. Within a bank, channels are arranged in 3-packs. GXB_L0 contains channels 0–2, GXB_L1 includes channels 3–5, and so on.

Figure 4-36: Cyclone V GX/GT/ST/ST Devices with 9 or 12 Transceiver Channels and 2 PCIe Cores

In the following figure, the Hard IP for PCI Express uses channel 1 and channel 2 of GXB_L0 and channel 1 and channel 1 of GXB_L2.

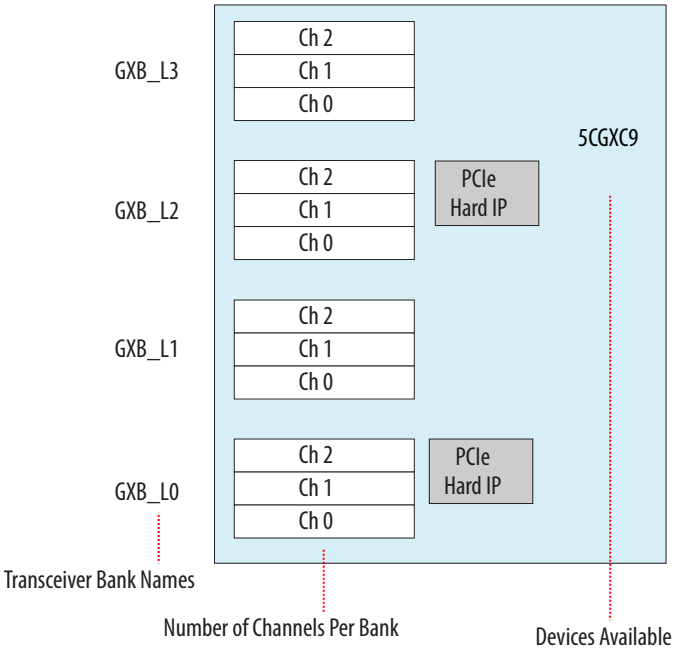
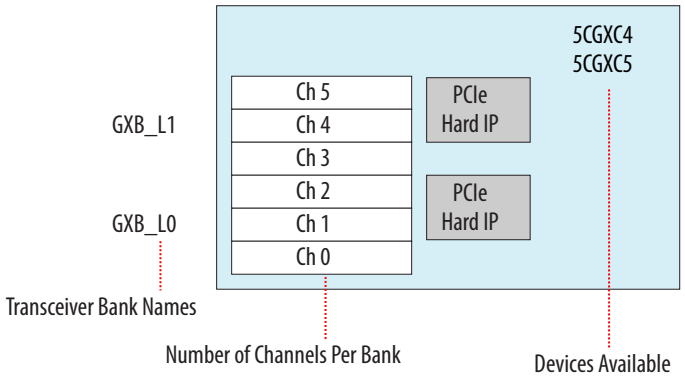


Figure 4-37: Cyclone V GX/GT/ST/ST Devices with 6 Transceiver Channels and 2 PCIe Cores



For more comprehensive information about Cyclone V transceivers, refer to the *Transceiver Banks* section in the *Transceiver Architecture in Cyclone V Devices*.

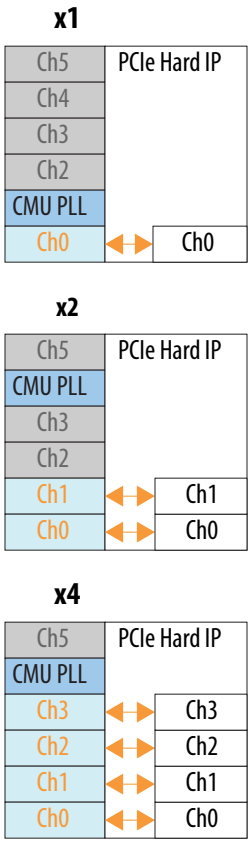
Related Information

Transceiver Architecture in Cyclone V Devices

Channel Placement in Cyclone V Devices

Figure 4-38: Cyclone V Gen1 and Gen2 Channel Placement Using the CMU PLL

In the following figures the channels shaded in blue provide the transmit CMU PLL generating the high-speed serial clock.



You can assign other protocols to unused channels the if data rate and clock specification exactly match the PCIe configuration.

PIPE Interface Signals

These PIPE signals are available for Gen1, Gen2, and Gen3 variants so that you can simulate using either the serial or the PIPE interface. Simulation is much faster using the PIPE interface because the PIPE simulation bypasses the SERDES model . By default, the PIPE interface is 8 bits for Gen1 and Gen2 and 32 bits for Gen3. You can use the PIPE interface for simulation even though your actual design includes a serial interface to the internal transceivers. However, it is not possible to use the Hard IP PIPE interface in hardware, including probing these signals using SignalTap® II Embedded Logic Analyzer.

Note: The Altera Root Port BFM bypasses Gen3 Phase 2 and Phase 3 Equalization. However, Gen3 variants can perform Phase 2 and Phase 3 equalization if instructed by a third-party BFM.

In the following table, signals that include lane number 0 also exist for lanes 1-7.

Table 4-21: PIPE Interface Signals

Signal	Direction	Description
txdata0[7:0]	Output	Transmit data <*n*>. This bus transmits data on lane <*n*>.
txdatak0	Output	Transmit data control <*n*>. This signal serves as the control bit for txdata <*n*>.
txdataskip0	Output	For Gen3 operation. Allows the MAC to instruct the TX interface to ignore the TX data interface for one clock cycle. The following encodings are defined: • 1'b0: TX data is invalid • 1'b1: TX data is valid
tx_deemph0	Output	Transmit de-emphasis selection. The Cyclone V Hard IP for PCI Express sets the value for this signal based on the indication received from the other end of the link during the Training Sequences (TS). You do not need to change this value.
rxdata0[7:0] (2)	Input	Receive data <*n*>. This bus receives data on lane <*n*>.
rxdatak0 (2)	Input	Receive data <*n*>. This bus receives data on lane <*n*>. Bit 0 corresponds to the lowest-order byte of rxdata, and so on. A value of 0 indicates a data byte. A value of 1 indicates a control byte. For Gen1 and Gen2 only.
txdetectrx0	Output	Transmit detect receive <*n*>. This signal tells the PHY layer to start a receive detection operation or to begin loopback.
txelecidle	Output	Transmit electrical idle <*n*>. This signal forces the TX output to electrical idle.
txcompl0	Output	Transmit compliance <*n*>. This signal forces the running disparity to negative in Compliance Mode (negative COM character).
rxpolarity0	Output	Receive polarity <*n*>. This signal instructs the PHY layer to invert the polarity of the 8B/10B receiver decoding block.
powerdown0[1:0]	Output	Power down <*n*>. This signal requests the PHY to change its power state to the specified state (P0, P0s, P1, or P2).
tx_margin[2:0]	Output	Transmit V_{OD} margin selection. The value for this signal is based on the value from the Link Control 2 Register. Available for simulation only.

Signal	Direction	Description
txswing	Output	When asserted, indicates full swing for the transmitter voltage. When deasserted indicates half swing.
rxvalid0 (1)	Input	Receive valid <*n*>. This signal indicates symbol lock and valid data on rxdata<*n*> and rxdatak <*n*>.
phystatus0 (1)	Input	PHY status <*n*>. This signal communicates completion of several PHY requests.
rxelecidle0 (1)	Input	Receive electrical idle <*n*>. When asserted, indicates detection of an electrical idle.
rxstatus0[2:0] (1)	Input	Receive status <*n*>. This signal encodes receive status, including error codes for the receive data stream and receiver detection.
simu_mode_pipe	Input	When set to 1, the PIPE interface is in simulation mode.
sim_pipe_rate[1:0]	Output	The 2-bit encodings have the following meanings: • 2'b00: Gen1 rate (2.5 Gbps) • 2'b01: Gen2 rate (5.0 Gbps) • 2'b1X: Gen3 rate (8.0 Gbps)
sim_pipe_pclk_in	Input	This clock is used for PIPE simulation only, and is derived from the refclk. It is the PIPE interface clock used for PIPE mode simulation.
sim_pipe_pclk_out	Output	TX datapath clock to the BFM PHY. pclk_out is derived from refclk and provides the source synchronous clock for TX data from the PHY.
sim_pipe_clk250_out	Output	Used to generate pclk.
sim_pipe_clk500_out	Output	Used to generate pclk.
sim_pipe_ltssmstate0[4:0]	Input and Output	LTSSM state: The LTSSM state machine encoding defines the following states: • 5'b00000: Detect.Quiet • 5'b 00001: Detect.Active • 5'b00010: Polling.Active • 5'b 00011: Polling.Compliance • 5'b 00100: Polling.Configuration • 5'b00101: Polling.Speed • 5'b00110: config.LinkwidthsStart

Signal	Direction	Description
		• 5'b 00111: Config.Linkaccept • 5'b 01000: Config.Lanenumaccept • 5'b01001: Config.Lanenumwait • 5'b01010: Config.Complete • 5'b 01011: Config.Idle • 5'b01100: Recovery.Rcvlock • 5'b01101: Recovery.Rcvconfig • 5'b01110: Recovery.Idle • 5'b 01111: L0 • 5'b10000: Disable • 5'b10001: Loopback.Entry • 5'b10010: Loopback.Active • 5'b10011: Loopback.Exit • 5'b10100: Hot.Reset • 5'b10101: L0s • 5'b11001: L2.transmit.Wake • 5'b11010: Speed.Recovery • 5'b11011: Recovery.Equalization, Phase 0 • 5'b11100: Recovery.Equalization, Phase 1 • 5'b11101: Recovery.Equalization, Phase 2 • 5'b11110: Recovery.Equalization, Phase 3 • 5'b11111: Recovery.Equalization, Done
`rxfreqlocked0` (1)	Input	When asserted indicates that the `pclk_in` used for PIPE simulation is valid.
`eidleinfersel0[2:0]`	Output	Electrical idle entry inference mechanism selection. The following encodings are defined: • 3'b0xx: Electrical Idle Inference not required in current LTSSM state • 3'b100: Absence of COM/SKP Ordered Set in the 128 us window for Gen1 or Gen2 • 3'b101: Absence of TS1/TS2 Ordered Set in a 1280 UI interval for Gen1 or Gen2 • 3'b110: Absence of Electrical Idle Exit in 2000 UI interval for Gen1 and 16000 UI interval for Gen2 • 3'b111: Absence of Electrical idle exit in 128 us window for Gen1

Notes:

1. These signals are for simulation only. For Quartus II software compilation, these pipe signals can be left floating.

Test Signals

Table 4-22: Test Interface Signals

The `test_in` bus provides run-time control and monitoring of the internal state of the IP core.

Signal	Direction	Description
`test_in[31:0]`	Input	The bits of the `test_in` bus have the following definitions: • [0]: Simulation mode. This signal can be set to 1 to accelerate initialization by reducing the value of many initialization counters. • [1]: Reserved. Must be set to 1'b0. • [2]: Descramble mode disable. This signal must be set to 1 during initialization in order to disable data scrambling. You can use this bit in simulation for both Endpoints and Root Ports to observe descrambled data on the link. Descrambled data cannot be used in open systems because the link partner typically scrambles the data. • [4:3]: Reserved. Must be set to 4'b01. • [5]: Compliance test mode. Disable/force compliance mode. When set, prevents the LTSSM from entering compliance mode. Toggling this bit controls the entry and exit from the compliance state, enabling the transmission of compliance patterns. • [6]: Forces entry to compliance mode when a timeout is reached in the polling.active state and not all lanes have detected their exit condition. • [7]: Disable low power state negotiation. Altera recommends setting thist bit. • [31:8] Reserved. Set to all 0s.
`simu_mode_pipe`	Input	When high, indicates that the PIPE interface is in simulation mode.
`testin_zero`	Output	When asserted, indicates accelerated initialization for simulation is active.

Registers 5

2014.12.15

UG-01110_avst

Subscribe Send Feedback

Correspondence between Configuration Space Registers and the PCIe Specification

Table 5-1: Correspondence between Configuration Space Capability Structures and PCIe Base Specification Description

For the Type 0 and Type 1 Configuration Space Headers, the first line of each entry lists Type 0 values and the second line lists Type 1 values when the values differ.

Byte Address	Hard IP Configuration Space Register	Corresponding Section in PCIe Specification
0x000:0x03C	PCI Header Type 0 Configuration Registers	Type 0 Configuration Space Header
0x000:0x03C	PCI Header Type 1 Configuration Registers	Type 1 Configuration Space Header
0x040:0x04C	Reserved	N/A
0x050:0x05C	MSI Capability Structure	MSI Capability Structure
0x068:0x070	MSI-X Capability Structure	MSI-X Capability Structure
0x070:0x074	Reserved	N/A
0x078:0x07C	Power Management Capability Structure	PCI Power Management Capability Structure
0x080:0x0B8	PCI Express Capability Structure	PCI Express Capability Structure
0x0B8:0x0FC	Reserved	N/A
0x094:0x0FF	Root Port	N/A
0x100:0x16C	Virtual Channel Capability Structure (Reserved)	Virtual Channel Capability

© 2014 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

Byte Address	Hard IP Configuration Space Register	Corresponding Section in PCIe Specification
0x170:0x17C	Reserved	N/A
0x180:0x1FC	Virtual channel arbitration table (Reserved)	VC Arbitration Table
0x200:0x23C	Port VC0 arbitration table (Reserved)	Port Arbitration Table
0x240:0x27C	Port VC1 arbitration table (Reserved)	Port Arbitration Table
0x280:0x2BC	Port VC2 arbitration table (Reserved)	Port Arbitration Table
0x2C0:0x2FC	Port VC3 arbitration table (Reserved)	Port Arbitration Table
0x300:0x33C	Port VC4 arbitration table (Reserved)	Port Arbitration Table
0x340:0x37C	Port VC5 arbitration table (Reserved)	Port Arbitration Table
0x380:0x3BC	Port VC6 arbitration table (Reserved)	Port Arbitration Table
0x3C0:0x3FC	Port VC7 arbitration table (Reserved)	Port Arbitration Table
0x400:0x7FC	Reserved	PCIe spec corresponding section name
0x800:0x834	Advanced Error Reporting AER (optional)	Advanced Error Reporting Capability
0x838:0xFFF	Reserved	N/A
0x000	Device ID, Vendor ID	Type 0 Configuration Space Header Type 1 Configuration Space Header
0x004	Status, Command	Type 0 Configuration Space Header Type 1 Configuration Space Header
0x008	Class Code, Revision ID	Type 0 Configuration Space Header Type 1 Configuration Space Header
0x00C	BIST, Header Type, Primary Latency Timer, Cache Line Size	Type 0 Configuration Space Header Type 1 Configuration Space Header
0x010	Base Address 0	Base Address Registers
0x014	Base Address 1	Base Address Registers

Byte Address	Hard IP Configuration Space Register	Corresponding Section in PCIe Specification
0x018	Base Address 2 Secondary Latency Timer, Subordinate Bus Number, Secondary Bus Number, Primary Bus Number	Base Address Registers Secondary Latency Timer, Type 1 Configuration Space Header, Primary Bus Number
0x01C	Base Address 3 Secondary Status, I/O Limit, I/O Base	Base Address Registers Secondary Status Register ,Type 1 Configuration Space Header
0x020	Base Address 4 Memory Limit, Memory Base	Base Address Registers Type 1 Configuration Space Header
0x024	Base Address 5 Prefetchable Memory Limit, Prefetchable Memory Base	Base Address Registers Prefetchable Memory Limit, Prefetchable Memory Base
0x028	Reserved Prefetchable Base Upper 32 Bits	N/A Type 1 Configuration Space Header
0x02C	Subsystem ID, Subsystem Vendor ID Prefetchable Limit Upper 32 Bits	Type 0 Configuration Space Header Type 1 Configuration Space Header
0x030	Expansion ROM base address I/O Limit Upper 16 Bits, I/O Base Upper 16 Bits	Type 0 Configuration Space Header Type 1 Configuration Space Header
0x034	Reserved, Capabilities PTR	Type 0 Configuration Space Header Type 1 Configuration Space Header
0x038	Reserved Expansion ROM Base Address	N/A Type 1 Configuration Space Header
0x03C	Interrupt Pin, Interrupt Line Bridge Control, Interrupt Pin, Interrupt Line	Type 0 Configuration Space Header Type 1 Configuration Space Header
0x050	MSI-Message Control Next Cap Ptr Capability ID	MSI and MSI-X Capability Structures
0x054	Message Address	MSI and MSI-X Capability Structures

Byte Address	Hard IP Configuration Space Register	Corresponding Section in PCIe Specification
0x058	Message Upper Address	MSI and MSI-X Capability Structures
0x05C	Reserved Message Data	MSI and MSI-X Capability Structures
0x068	MSI-X Message Control Next Cap Ptr Capability ID	MSI and MSI-X Capability Structures
0x06C	MSI-X Table Offset BIR	MSI and MSI-X Capability Structures
0x070	Pending Bit Array (PBA) Offset BIR	MSI and MSI-X Capability Structures
0x078	Capabilities Register Next Cap PTR Cap ID	PCI Power Management Capability Structure
0x07C	Data PM Control/Status Bridge Extensions Power Management Status & Control	PCI Power Management Capability Structure
0x800	PCI Express Enhanced Capability Header	Advanced Error Reporting Enhanced Capability Header
0x804	Uncorrectable Error Status Register	Uncorrectable Error Status Register
0x808	Uncorrectable Error Mask Register	Uncorrectable Error Mask Register
0x80C	Uncorrectable Error Severity Register	Uncorrectable Error Severity Register
0x810	Correctable Error Status Register	Correctable Error Status Register
0x814	Correctable Error Mask Register	Correctable Error Mask Register
0x818	Advanced Error Capabilities and Control Register	Advanced Error Capabilities and Control Register
0x81C	Header Log Register	Header Log Register
0x82C	Root Error Command	Root Error Command Register
0x830	Root Error Status	Root Error Status Register
0x834	Error Source Identification Register Correctable Error Source ID Register	Error Source Identification Register

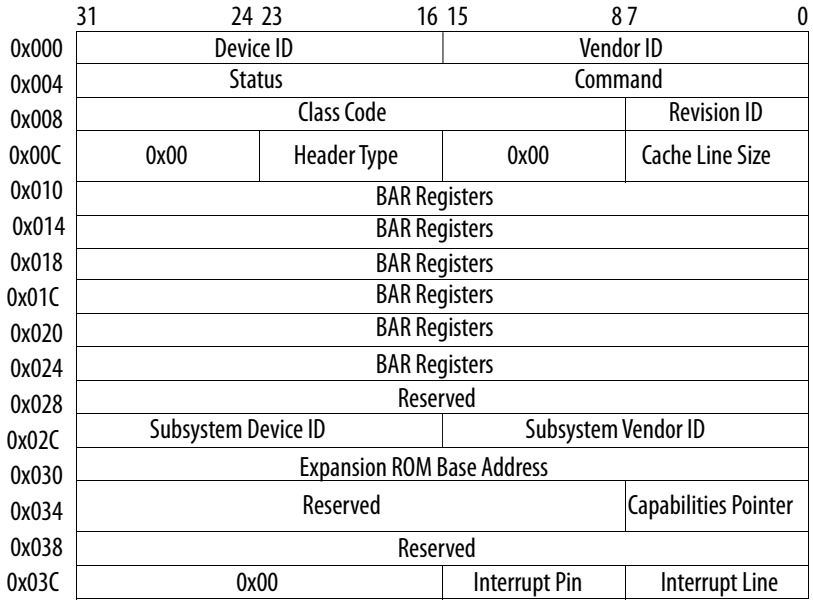
Related Information

PCI Express Base Specification 2.1 or 3.0

Type 0 Configuration Space Registers

Figure 5-1: Type 0 Configuration Space Registers - Byte Address Offsets and Layout

Endpoints store configuration data in the Type 0 Configuration Space. The **Correspondence between Configuration Space Registers and the PCIe Specification** on page 5-1 lists the appropriate section of the *PCI Express Base Specification* that describes these registers.



PCI Express Capability Structures

Figure 5-2: MSI Capability Structure

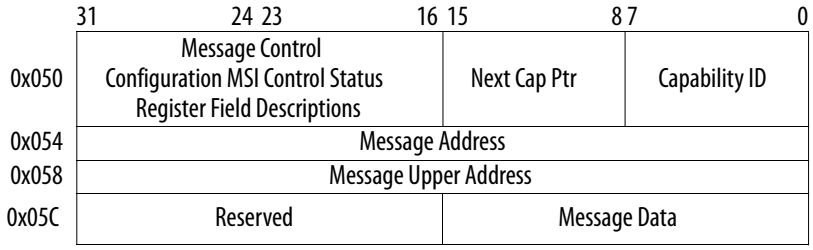


Figure 5-3: MSI-X Capability Structure

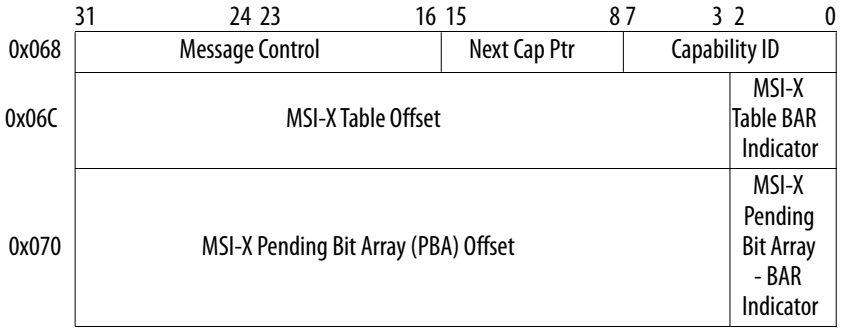


Figure 5-4: Power Management Capability Structure - Byte Address Offsets and Layout

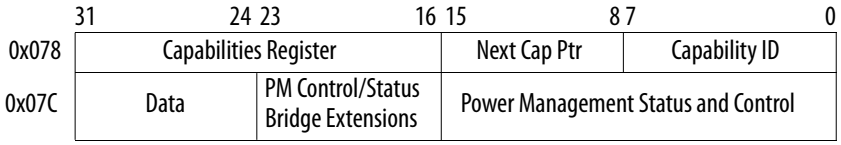
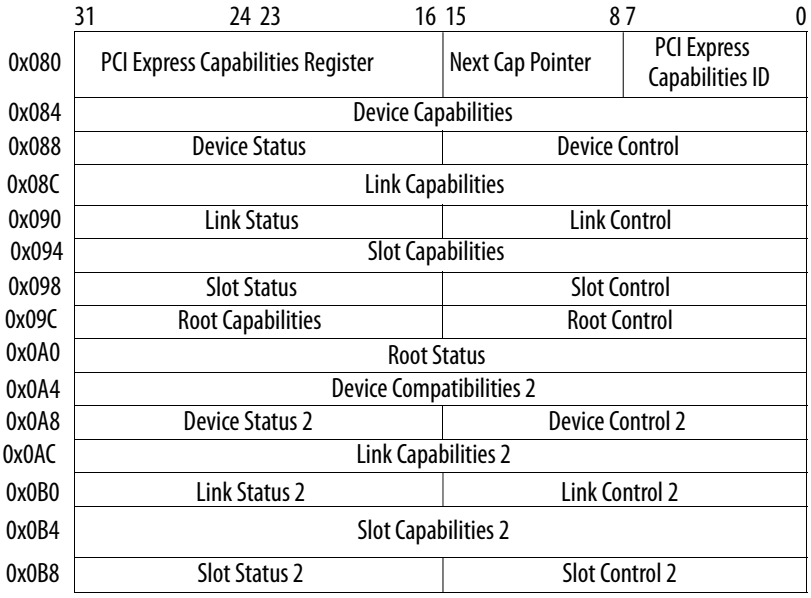


Figure 5-5: PCI Express AER Extended Capability Structure

Byte Offset	31:24	23:16	15:8	7:0
0x800	PCI Express Enhanced Capability Register			
0x804	Uncorrectable Error Status Register			
0x808	Uncorrectable Error Mask Register			
0x80C	Uncorrectable Error Severity Register			
0x810	Correctable Error Status Register			
0x814	Correctable Error Mask Register			
0x818	Advanced Error Capabilities and Control Register			
0x81C	Header Log Register			
0x82C	Root Error Command Register			
0x830	Root Error Status Register			
0x834	Error Source Identification Register		Correctable Error Source Identification Register	

Figure 5-6: PCI Express Capability Structure - Byte Address Offsets and Layout

In the following table showing the PCI Express Capability Structure, registers that are not applicable to a device are reserved.



Altera-Defined VSEC Registers

Figure 5-7: VSEC Registers

This extended capability structure supports Configuration via Protocol (CvP) programming and detailed internal error reporting.

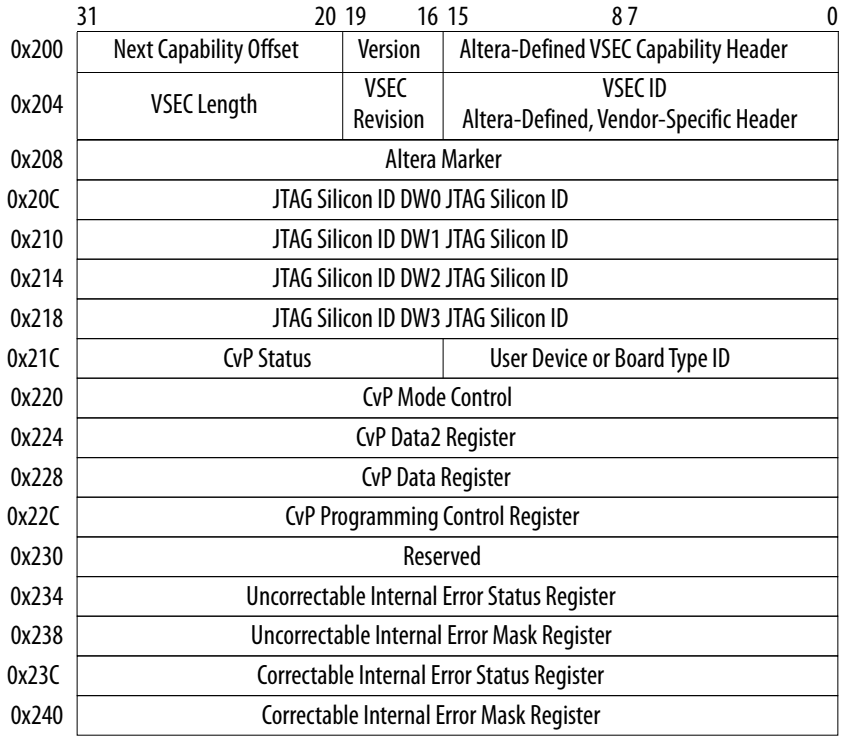


Table 5-2: Altera-Defined VSEC Capability Register, 0x200

The Altera-Defined Vendor Specific Extended Capability. This extended capability structure supports Configuration via Protocol (CvP) programming and detailed internal error reporting.

Bits	Register Description	Value	Access
[15:0]	PCI Express Extended Capability ID. Altera-defined value for VSEC Capability ID.	0x000B	RO
[19:16]	Version. Altera-defined value for VSEC version.	0x1	RO
[31:20]	Next Capability Offset. Starting address of the next Capability Structure implemented, if any.	Variable	RO

Table 5-3: Altera-Defined Vendor Specific Header

You can specify these values when you instantiate the Hard IP. These registers are read-only at run-time.

Bits	Register Description	Value	Access
[15:0]	`VSEC ID`. A user configurable VSEC ID.	User entered	RO
[19:16]	`VSEC Revision`. A user configurable VSEC revision.	Variable	RO
[31:20]	`VSEC Length`. Total length of this structure in bytes.	0x044	RO

Table 5-4: Altera Marker Register

Bits	Register Description	Value	Access
[31:0]	`Altera Marker`. This read only register is an additional marker. If you use the standard Altera Programmer software to configure the device with CvP, this marker provides a value that the programming software reads to ensure that it is operating with the correct VSEC.	A Device Value	RO

Table 5-5: JTAG Silicon ID Register

Bits	Register Description	Value	Access
[127:96]	`JTAG Silicon ID DW3`	Application Specific	RO
[95:64]	`JTAG Silicon ID DW2`	Application Specific	RO
[63:32]	`JTAG Silicon ID DW1`	Application Specific	RO
[31:0]	`JTAG Silicon ID DW0`. This is the JTAG Silicon ID that CvP programming software reads to determine that the correct SRAM object file (**.sof**) is being used.	Application Specific	RO

Table 5-6: User Device or Board Type ID Register

Bits	Register Description	Value	Access
[15:0]	Configurable device or board type ID to specify to CvP the correct **.sof**.	Variable	RO

CvP Registers

Table 5-7: CvP Status

The `CvP Status` register allows software to monitor the CvP status signals.

Bits	Register Description	Reset Value	Access
[31:26]	Reserved	0x00	RO
[25]	`PLD_CORE_READY`. From FPGA fabric. This status bit is provided for debug.	Variable	RO
[24]	`PLD_CLK_IN_USE`. From clock switch module to fabric. This status bit is provided for debug.	Variable	RO
[23]	`CVP_CONFIG_DONE`. Indicates that the FPGA control block has completed the device configuration via CvP and there were no errors.	Variable	RO
[22]	Reserved	Variable	RO
[21]	`USERMODE`. Indicates if the configurable FPGA fabric is in user mode.	Variable	RO
[20]	`CVP_EN`. Indicates if the FPGA control block has enabled CvP mode.	Variable	RO
[19]	`CVP_CONFIG_ERROR`. Reflects the value of this signal from the FPGA control block, checked by software to determine if there was an error during configuration.	Variable	RO
[18]	`CVP_CONFIG_READY`. Reflects the value of this signal from the FPGA control block, checked by software during programming algorithm.	Variable	RO
[17:0]	Reserved	Variable	RO

Table 5-8: CvP Mode Control

The `CvP Mode Control` register provides global control of the CvP operation.

Bits	Register Description	Reset Value	Access
[31:16]	Reserved.	0x0000	RO
[15:8]	`CVP_NUMCLKS`. This is the number of clocks to send for every CvP data write. Set this field to one of the values below depending on your configuration image: • 0x01 for uncompressed and unencrypted images • 0x04 for uncompressed and encrypted images • 0x08 for all compressed images	0x00	RW
[7:3]	Reserved.	0x0	RO
[2]	`CVP_FULLCONFIG`. Request that the FPGA control block reconfigure the entire FPGA including the Cyclone V Hard IP for PCI Express, bring the PCIe link down.	1'b0	RW

Bits	Register Description	Reset Value	Access
[1]	HIP_CLK_SEL. Selects between PMA and fabric clock when USER_MODE = 1 and PLD_CORE_READY = 1. The following encodings are defined: • 1: Selects internal clock from PMA which is required for CVP_MODE. • 0: Selects the clock from soft logic fabric. This setting should only be used when the fabric is configured in USER_MODE with a configuration file that connects the correct clock. To ensure that there is no clock switching during CvP, you should only change this value when the Hard IP for PCI Express has been idle for 10 µs and wait 10 µs after changing this value before resuming activity.	1'b0	RW
[0]	CVP_MODE. Controls whether the IP core is in CVP_MODE or normal mode. The following encodings are defined: • 1:CVP_MODE is active. Signals to the FPGA control block active and all TLPs are routed to the Configuration Space. This CVP_MODE cannot be enabled if CVP_EN = 0. • 0: The IP core is in normal mode and TLPs are routed to the FPGA fabric.	1'b0	RW

Table 5-9: CvP Data Registers

The following table defines the CvP Data registers. For 64-bit data, the optional CvP Data2 stores the upper 32 bits of data. Programming software should write the configuration data to these registers. If you Every write to these register sets the data output to the FPGA control block and generates <*n*> clock cycles to the FPGA control block as specified by the CVP_NUM_CLKS field in the CvP Mode Control register. Software must ensure that all bytes in the memory write dword are enabled. You can access this register using configuration writes, alternatively, when in CvP mode, these registers can also be written by a memory write to any address defined by a memory space BAR for this device. Using memory writes should allow for higher throughput than configuration writes.

Bits	Register Description	Reset Value	Access
[31:0]	Upper 32 bits of configuration data to be transferred to the FPGA control block to configure the device. You can choose 32- or 64-bit data.	0x00000000	RW
[31:0]	Lower 32 bits of configuration data to be transferred to the FPGA control block to configure the device.	0x00000000	RW

Table 5-10: CvP Programming Control Register

This register is written by the programming software to control CvP programming.

Bits	Register Description	Reset Value	Access
[31:2]	Reserved.	0x0000	RO

Bits	Register Description	Reset Value	Access
[1]	START_XFER. Sets the CvP output to the FPGA control block indicating the start of a transfer.	1'b0	RW
[0]	CVP_CONFIG. When asserted, instructs that the FPGA control block begin a transfer via CvP.	1'b0	RW

Uncorrectable Internal Error Mask Register

Table 5-11: Uncorrectable Internal Error Mask Register

The Uncorrectable Internal Error Mask register controls which errors are forwarded as internal uncorrectable errors. With the exception of the configuration error detected in CvP mode, all of the errors are severe and may place the device or PCIe link in an inconsistent state. The configuration error detected in CvP mode may be correctable depending on the design of the programming software. The access code *RWS* stands for Read Write Sticky meaning the value is retained after a soft reset of the IP core.

Bits	Register Description	Reset Value	Access
[31:12]	Reserved.	1b'0	RO
[11]	Mask for RX buffer posted and completion overflow error.	1b'1	RWS
[10]	Reserved	1b'0	RO
[9]	Mask for parity error detected on Configuration Space to TX bus interface.	1b'1	RWS
[8]	Mask for parity error detected on the TX to Configuration Space bus interface.	1b'1	RWS
[7]	Mask for parity error detected at TX Transaction Layer error.	1b'1	RWS
[6]	Reserved	1b'0	RO
[5]	Mask for configuration errors detected in CvP mode.	1b'0	RWS
[4]	Mask for data parity errors detected during TX Data Link LCRC generation.	1b'1	RWS
[3]	Mask for data parity errors detected on the RX to Configuration Space Bus interface.	1b'1	RWS
[2]	Mask for data parity error detected at the input to the RX Buffer.	1b'1	RWS
[1]	Mask for the retry buffer uncorrectable ECC error.	1b'1	RWS

Bits	Register Description	Reset Value	Access
[0]	Mask for the RX buffer uncorrectable ECC error.	1b'1	RWS

Uncorrectable Internal Error Status Register

Table 5-12: Uncorrectable Internal Error Status Register

This register reports the status of the internally checked errors that are uncorrectable. When specific errors are enabled by the `Uncorrectable Internal Error Mask` register, they are handled as Uncorrectable Internal Errors as defined in the *PCI Express Base Specification 3.0*. This register is for debug only. It should only be used to observe behavior, not to drive custom logic. The access code RW1CS represents Read Write 1 to Clear Sticky.

Bits	Register Description	Reset Value	Access
[31:12]	Reserved.	0	RO
[11]	When set, indicates an RX buffer overflow condition in a posted request or Completion	0	RW1CS
[10]	Reserved.	0	RO
[9]	When set, indicates a parity error was detected on the Configuration Space to TX bus interface	0	RW1CS
[8]	When set, indicates a parity error was detected on the TX to Configuration Space bus interface	0	RW1CS
[7]	When set, indicates a parity error was detected in a TX TLP and the TLP is not sent.	0	RW1CS
[6]	When set, indicates that the Application Layer has detected an uncorrectable internal error.	0	RW1CS
[5]	When set, indicates a configuration error has been detected in CvP mode which is reported as uncorrectable. This bit is set whenever a `CVP_CONFIG_ERROR` rises while in `CVP_MODE`.	0	RW1CS
[4]	When set, indicates a parity error was detected by the TX Data Link Layer.	0	RW1CS
[3]	When set, indicates a parity error has been detected on the RX to Configuration Space bus interface.	0	RW1CS
[2]	When set, indicates a parity error was detected at input to the RX Buffer.	0	RW1CS

Bits	Register Description	Reset Value	Access
[1]	When set, indicates a retry buffer uncorrectable ECC error.	0	RW1CS
[0]	When set, indicates a RX buffer uncorrectable ECC error.	0	RW1CS

Correctable Internal Error Mask Register

Table 5-13: Correctable Internal Error Mask Register

The `Correctable Internal Error Mask` register controls which errors are forwarded as Internal Correctable Errors. This register is for debug only.

Bits	Register Description	Reset Value	Access
[31:7]	Reserved.	0	RO
[6]	Mask for Corrected Internal Error reported by the Application Layer.	1	RWS
[5]	Mask for configuration error detected in CvP mode.	0	RWS
[4:2]	Reserved.	0	RO
[1]	Mask for retry buffer correctable ECC error.	1	RWS
[0]	Mask for RX Buffer correctable ECC error.	1	RWS

Correctable Internal Error Status Register

Table 5-14: Correctable Internal Error Status Register

The `Correctable Internal Error Status` register reports the status of the internally checked errors that are correctable. When these specific errors are enabled by the `Correctable Internal Error Mask` register, they are forwarded as Correctable Internal Errors as defined in the *PCI Express Base Specification 3.0*. This register is for debug only. It should only be used to observe behavior, not to drive logic custom logic.

Bits	Register Description	Reset Value	Access
[31:6]	Reserved.	0	RO
[5]	When set, indicates a configuration error has been detected in CvP mode which is reported as correctable. This bit is set whenever a `CVP_CONFIG_ERROR` occurs while in `CVP_MODE`.	0	RW1CS

Bits	Register Description	Reset Value	Access
[4:2]	Reserved.	0	RO
[1]	When set, the retry buffer correctable ECC error status indicates an error.	0	RW1CS
[0]	When set, the RX buffer correctable ECC error status indicates an error.	0	RW1CS

Reset and Clocks

6

2014.12.15

UG-01110_avst

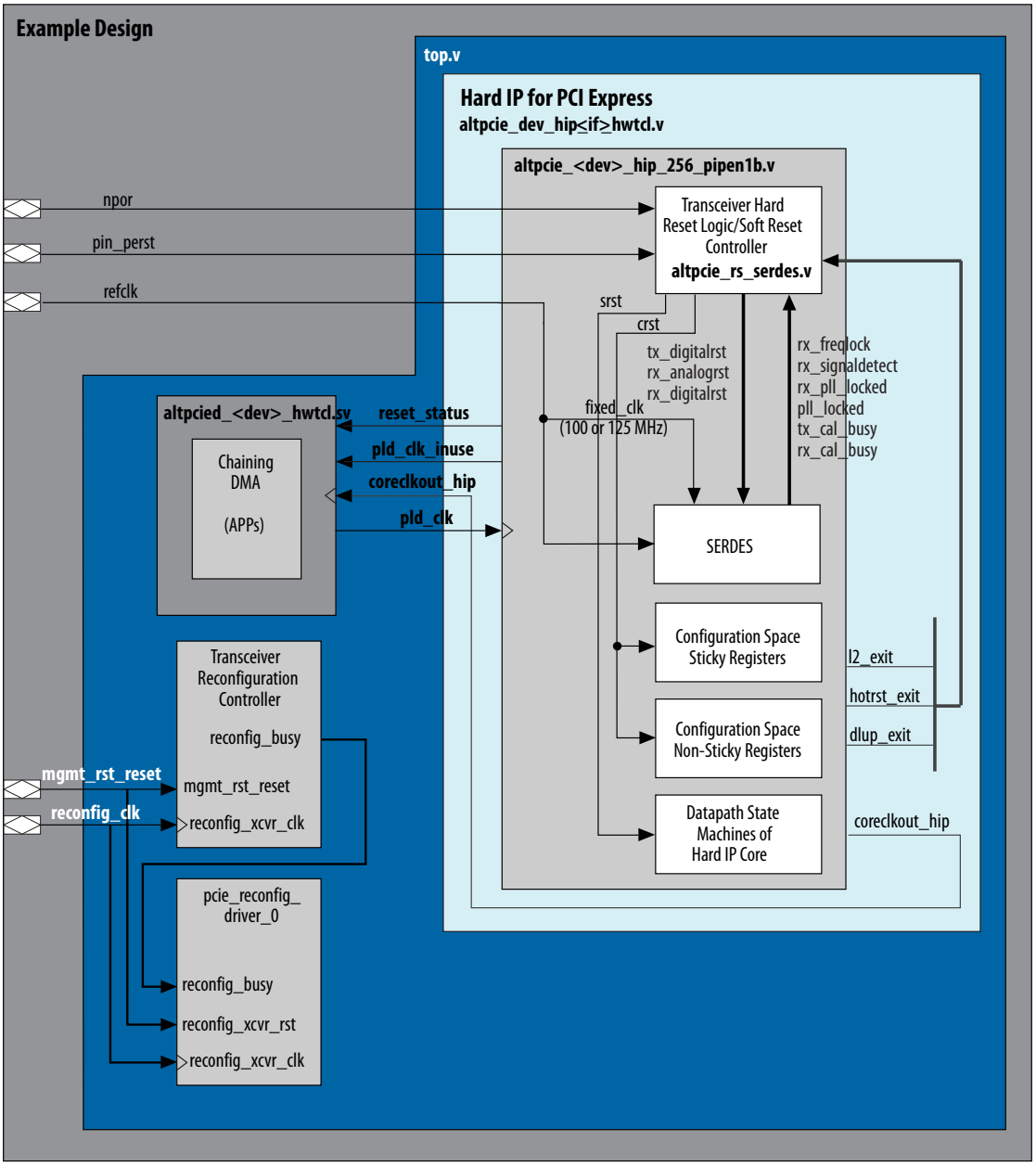
Subscribe Send Feedback

The `pin_perst` signal from the input pin of the FPGA resets the Hard IP for PCI Express IP Core. `app_rstn` which resets the Application Layer logic is derived from `reset_status` and `pld_clk_inuse`, which are outputs of the core. This reset controller is implemented in hardened logic. The figure below provides a simplified view of the logic that implements the reset controller.

© 2014 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

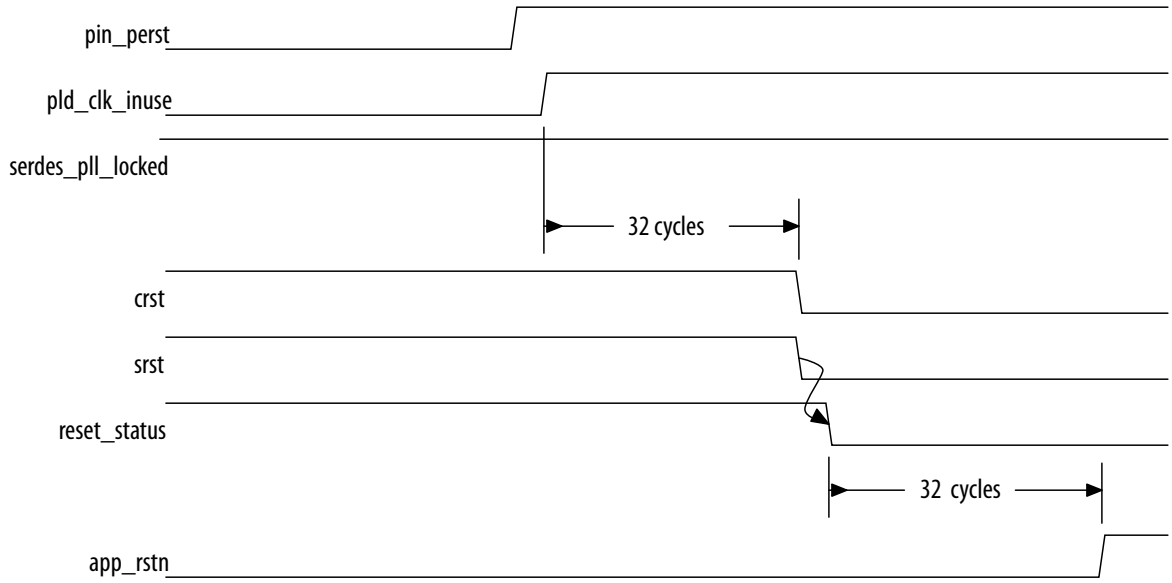
Figure 6-1: Reset Controller Block Diagram



Reset Sequence for Hard IP for PCI Express IP Core and Application Layer

Figure 6-2: Hard IP for PCI Express and Application Logic Reset Sequence

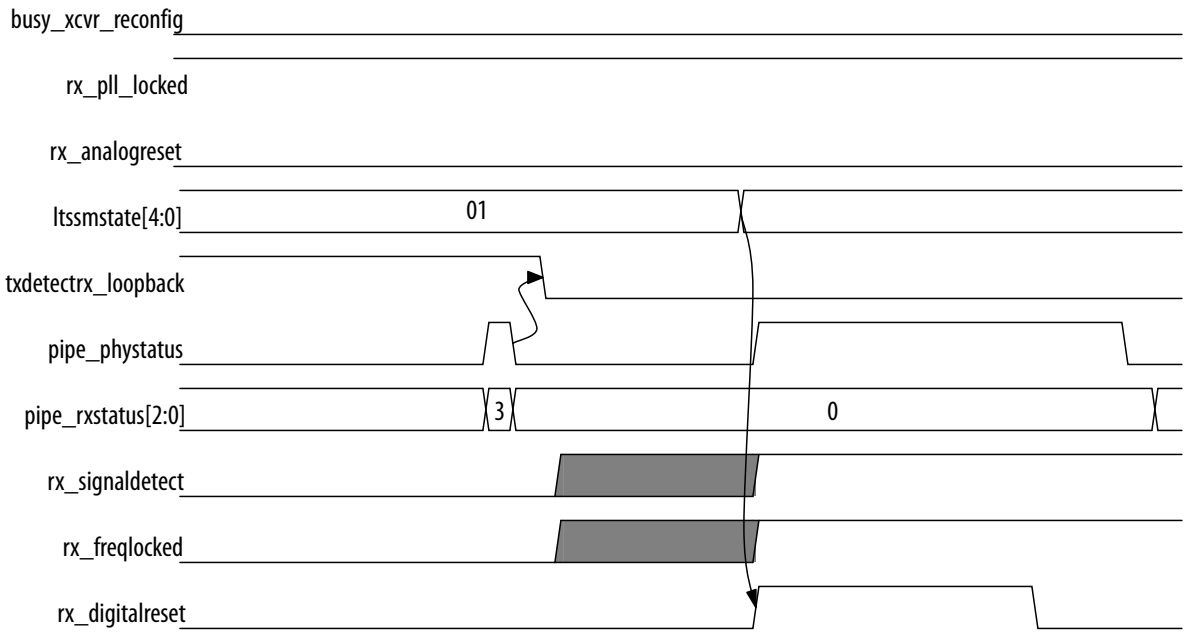
Your Application Layer can instantiate a module similar to the one in this figure to generate `app_rstn`, which resets the Application Layer logic.



This reset sequence includes the following steps:

1. After `pin_perst` or `npor` is released, the Hard IP reset controller waits for `pld_clk_inuse` to be asserted.
2. `csrt` and `srst` are released 32 cycles after `pld_clk_inuse` is asserted.
3. The Hard IP for PCI Express deasserts the `reset_status` output to the Application Layer.
4. The **altpcied_<*device*>v_hwtcl.sv** deasserts `app_rstn` 32 `pld_clk`cycles after `reset_status` is released.

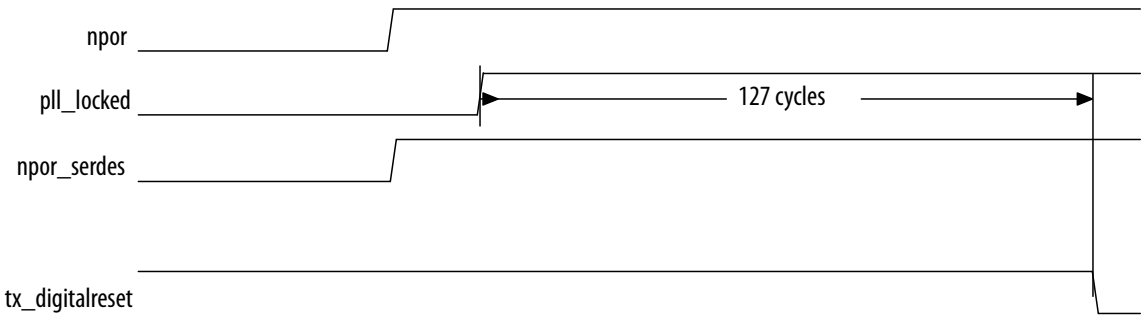
Figure 6-3: RX Transceiver Reset Sequence



The RX transceiver reset sequence includes the following steps:

1. After `rx_pll_locked` is asserted, the LTSSM state machine transitions from the Detect.Quiet to the Detect.Active state.
2. When the `pipe_phystatus` pulse is asserted and `pipe_rxstatus[2:0]` = 3, the receiver detect operation has completed.
3. The LTSSM state machine transitions from the Detect.Active state to the Polling.Active state.
4. The Hard IP for PCI Express asserts `rx_digitalreset`. The `rx_digitalreset` signal is deasserted after `rx_signaldetect` is stable for a minimum of 3 ms.

Figure 6-4: TX Transceiver Reset Sequence



The TX transceiver reset sequence includes the following steps:

1. After `npor` is deasserted, the IP core deasserts the `npor_serdes` input to the TX transceiver.
2. The SERDES reset controller waits for `pll_locked` to be stable for a minimum of 127 `pld_clk` cycles before deasserting `tx_digitalreset`.

For descriptions of the available reset signals, refer to *Reset Signals, Status, and Link Training Signals*.

Related Information

Reset, Status, and Link Training Signals on page 4-24

Clocks

The Hard IP contains a clock domain crossing (CDC) synchronizer at the interface between the PHY/MAC and the DLL layers. The synchronizer allows the Data Link and Transaction Layers to run at frequencies independent of the PHY/MAC. The CDC synchronizer provides more flexibility for the user clock interface. Depending on parameters you specify, the core selects the appropriate `coreclkout_hip`. You can use these parameters to enhance performance by running at a higher frequency for latency optimization or at a lower frequency to save power.

In accordance with the *PCI Express Base Specification*, you must provide a 100 MHz reference clock that is connected directly to the transceiver.

As a convenience, you may also use a 125 MHz input reference clock as input to the TX PLL.

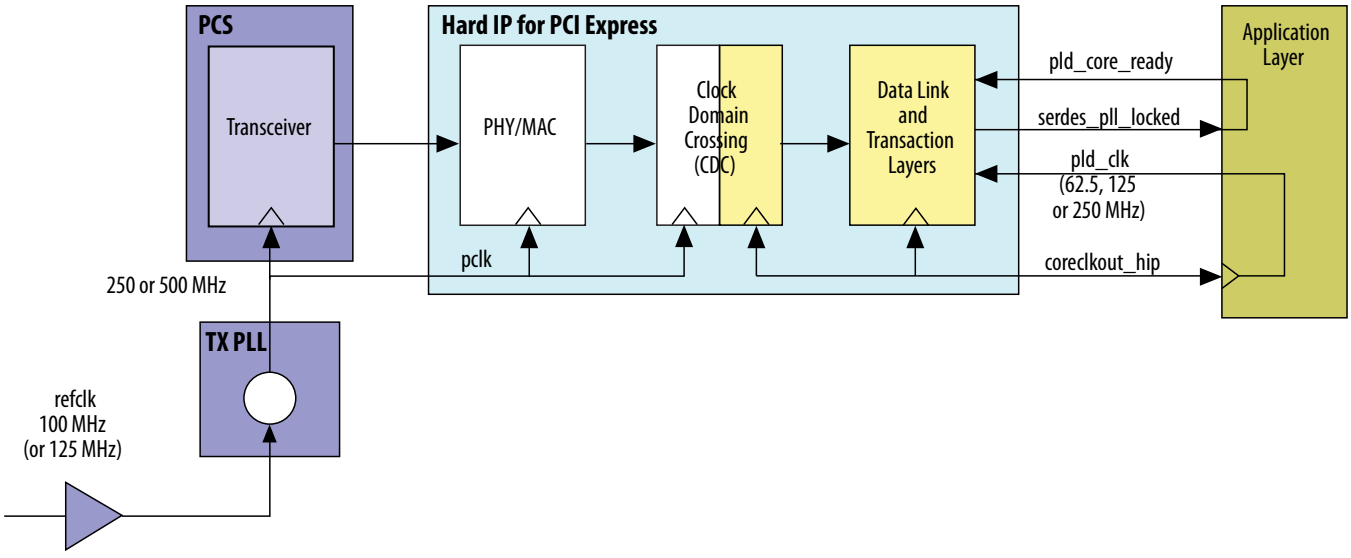
Related Information

PCI Express Base Specification 2.1 or 3.0

Clock Domains

Figure 6-5: Clock Domains and Clock Generation for the Application Layer

The following illustrates the clock domains when using `coreclkout_hip` to drive the Application Layer and the `pld_clk` of the IP core. The Altera-provided example design connects `coreclkout_hip` to the `pld_clk`. However, this connection is not mandatory.



As this figure indicates, the IP core includes the following clock domains:

pclk

The transceiver derives pclk from the 100 MHz refclk signal that you must provide to the device.

The *PCI Express Base Specification* requires that the refclk signal frequency be 100 MHz ±300 PPM.

The transitions between Gen1 and Gen2 should be glitchless. pclk can be turned off for most of the 1 ms timeout assigned for the PHY to change the clock rate; however, pclk should be stable before the 1 ms timeout expires.

Table 6-1: pclk Clock Frequency

Data Rate	Frequency
Gen1	250 MHz
Gen2	500 MHz

The CDC module implements the asynchronous clock domain crossing between the PHY/MAC pclk domain and the Data Link Layer coreclk domain. The transceiver pclk clock is connected directly to the Hard IP for PCI Express and does not connect to the FPGA fabric.

Related Information

PCI Express Base Specification 2.1 or 3.0

coreclkout_hip

Table 6-2: Application Layer Clock Frequency for All Combinations of Link Width, Data Rate and Application Layer Interface Widths

The coreclkout_hip signal is derived from pclk. The following table lists frequencies for coreclkout_hip, which are a function of the link width, data rate, and the width of the Application Layer to Transaction Layer interface. The frequencies and widths specified in this table are maintained throughout operation. If the link downtrains to a lesser link width or changes to a different maximum link rate, it maintains the frequencies it was originally configured for as specified in this table. (The Hard IP throttles the interface to achieve a lower throughput.)

Link Width	Maximum Link Rate	Avalon Interface Width	coreclkout_hip
×1	Gen1	64	62.5 MHz(5)
×1	Gen1	64	125 MHz
×2	Gen1	64	125 MHz
×4	Gen1	64	125 MHz

(5) This mode saves power

Link Width	Maximum Link Rate	Avalon Interface Width	coreclkout_hip
×8	Gen1	128	125 MHz
×1	Gen2	64	125 MHz
×2	Gen2	64	125 MHz
×4	Gen2	128	125 MHz

pld_clk

`coreclkout_hip` can drive the Application Layer clock along with the `pld_clk` input to the IP core. The `pld_clk` can optionally be sourced by a different clock than `coreclkout_hip`. The `pld_clk` minimum frequency cannot be lower than the `coreclkout_hip` frequency. Based on specific Application Layer constraints, a PLL can be used to derive the desired frequency.

Clock Summary

Table 6-3: Clock Summary

Name	Frequency	Clock Domain
`coreclkout_hip`	62.5, 125 or 250 MHz	Avalon-ST interface between the Transaction and Application Layers.
`pld_clk`	62.5, 125, or 250 MHz	Application and Transaction Layers.
`refclk`	100 or 125 MHz	SERDES (transceiver). Dedicated free running input clock to the SERDES block.
`reconfig_xcvr_clk`	100 –125 MHz	Transceiver Reconfiguration Controller.
`hip_reconfig_clk`	50–125 MHz	Avalon-MM interface for Hard IP dynamic reconfiguration interface which you can use to change the value of read-only configuration registers at run-time. This interface is optional. It is not required for Arria 10 devices.

Interrupts 7

2014.12.15

UG-01110_avst

Subscribe Send Feedback

Interrupts for Endpoints

The Cyclone V Hard IP for PCI Express provides support for PCI Express MSI, MSI-X, and legacy interrupts when configured in Endpoint mode. The MSI, MSI-X, and legacy interrupts are *mutually exclusive.* After power up, the Hard IP block starts in legacy interrupt mode, after which time software decides whether to switch to MSI mode by programming the `msi_enable` bit of the `MSI Message Control Register`, (bit[16] of 0x050) to 1, or to MSI-X mode if you turn on **Implement MSI-X** under the **PCI Express/PCI Capabilities** tab using the parameter editor. If you turn on the **Implement MSI-X** option, you should implement the MSI-X table structures at the memory space pointed to by the BARs.

Refer to section 6.1 of *PCI Express Base Specification* for a general description of PCI Express interrupt support for Endpoints.

Related Information

PCI Express Base Specification 2.1 or 3.0

MSI Interrupts

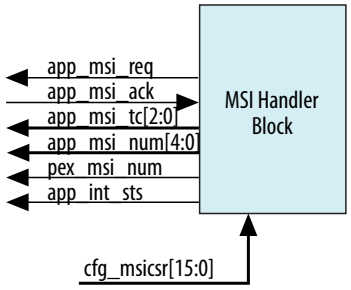
MSI interrupts are signaled on the PCI Express link using a single dword memory write TLP generated internally by the Cyclone V Hard IP for PCI Express. The `app_msi_req` input port controls MSI interrupt generation. When the input port asserts `app_msi_req`, it causes a MSI posted write TLP to be generated based on the MSI configuration register values and the `app_msi_tc` (traffic class) and `app_msi_num` (number) input ports. To enable MSI interrupts, software must first set the `MSI enable` bit and then disable legacy interrupts by setting the `Interrupt Disable` which is bit 10 of the `Command` register.

The following figure illustrates the architecture of the MSI handler block.

© 2014 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

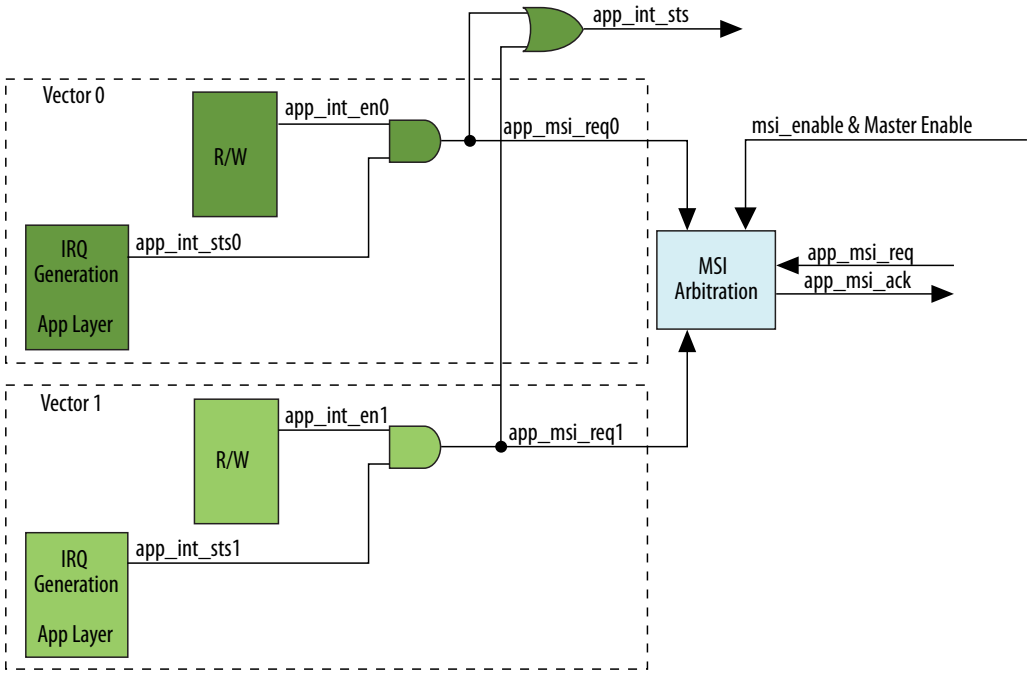
ISO 9001:2008 Registered

Figure 7-1: MSI Handler Block



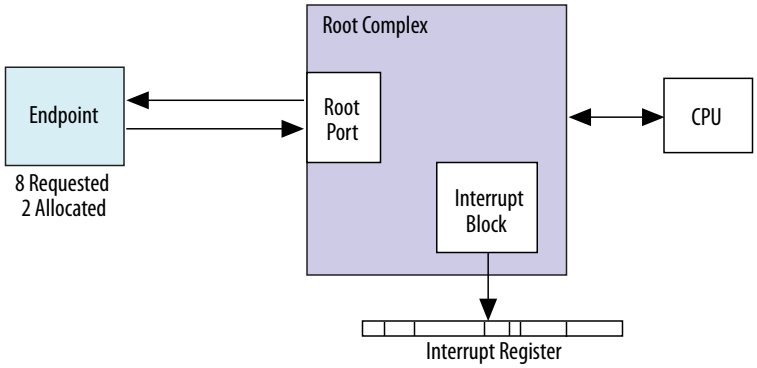
The following figure illustrates a possible implementation of the MSI handler block with a per vector enable bit. A global Application Layer interrupt enable can also be implemented instead of this per vector MSI.

Figure 7-2: Example Implementation of the MSI Handler Block



There are 32 possible MSI messages. The number of messages requested by a particular component does not necessarily correspond to the number of messages allocated. For example, in the following figure, the Endpoint requests eight MSIs but is only allocated two. In this case, you must design the Application Layer to use only two allocated messages.

Figure 7-3: MSI Request Example



The following table describes three example implementations. The first example allocates all 32 MSI messages. The second and third examples only allocate 4 interrupts.

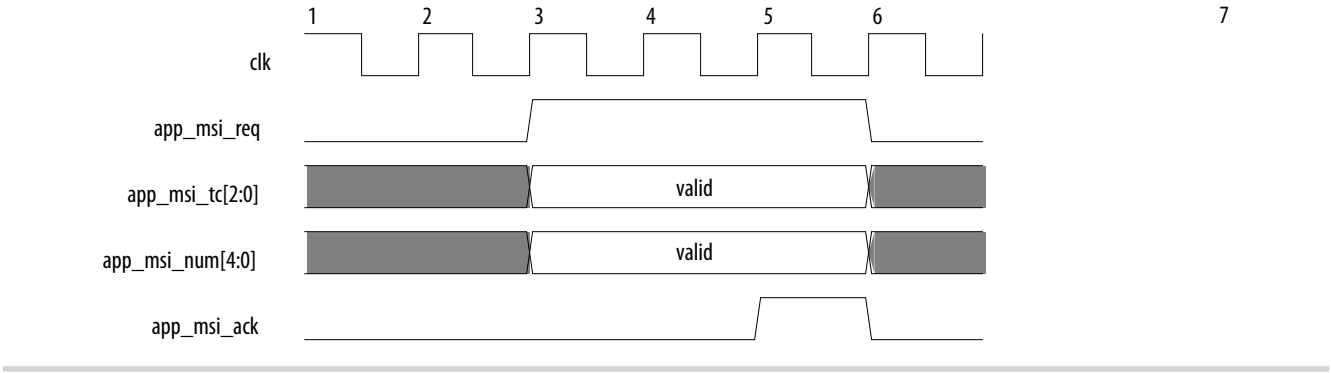
Table 7-1: MSI Messages Requested, Allocated, and Mapped

MSI	Allocated		
	32	4	4
System Error	31	3	3
Hot Plug and Power Management Event	30	2	3
Application Layer	29:0	1:0	2:0

MSI interrupts generated for Hot Plug, Power Management Events, and System Errors always use Traffic Class 0. MSI interrupts generated by the Application Layer can use any Traffic Class. For example, a DMA that generates an MSI at the end of a transmission can use the same traffic control as was used to transfer data.

The following figure illustrates the interactions among MSI interrupt signals for the Root Port. The minimum latency possible between `app_msi_req` and `app_msi_ack` is one clock cycle. In this timing diagram `app_msi_req` can extend beyond `app_msi_ack` before deasserting. However, `app_msi_req` must be deasserted before or within the same clock as `app_msi_ack` is deasserted to avoid inferring a new interrupt.

Figure 7-4: MSI Interrupt Signals Timing



Related Information

Correspondence between Configuration Space Registers and the PCIe Specification on page 5-1

MSI-X

You can enable MSI-X interrupts by turning on **Implement MSI-X** under the **PCI Express/PCI Capabilities** heading using the parameter editor. If you turn on the **Implement MSI-X** option, you should implement the MSI-X table structures at the memory space pointed to by the BARs as part of your Application Layer.

MSI-X TLPs are generated by the Application Layer and sent through the TX interface. They are single dword memory writes so that `Last DW Byte Enable` in the TLP header must be set to 4b'0000. MSI-X TLPs should be sent only when enabled by the MSI-X enable and the function mask bits in the message control for MSI-X Configuration register. These bits are available on the `tl_cfg_ctl` output bus.

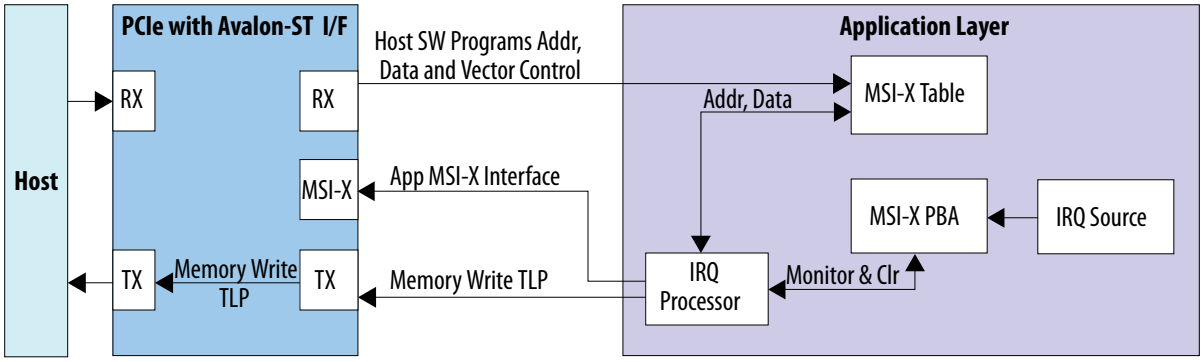
Related Information

- **PCI Express Base Specification 2.1 or 3.0**
- **PCI Local Bus Specification**

Implementing MSI-X Interrupts

Section 6.8.2 of the *PCI Local Bus Specification* describes the MSI-X capability and table structures. The MSI-X capability structure points to the MSI-X Table structure and MSI-X Pending Bit Array (PBA) registers. The BIOS sets up the starting address offsets and BAR associated with the pointer to the starting address of the MSI-X table and PBA registers.
The following figure shows the Application Layer modules that implement MSI-X interrupts.

Figure 7-5: MSI-X Interrupt Components



1. Host software sets up the MSI-X interrupts in the Application Layer by completing the following steps:
   a. Host software reads the `Message Control` register at 0x050 register to determine the MSI-X Table size. The number of table entries is the <*value read*> + *1*.

      The maximum table size is 2048 entries. Each 16-byte entry is divided in 4 fields as shown in the figure below. The MSI-X table can reside in any BAR. The base address of the MSI-X table must be aligned to a 4 KByte boundary.
   b. The host sets up the MSI-X table. It programs MSI-X address, data, and masks bits for each entry as shown in the figure below.

Figure 7-6: Format of MSI-X Table

DWORD 3	DWORD 2	DWORD 1	DWORD 0		Host Byte Addresses
Vector Control	Message Data	Message Upper Address	Message Address	Entry 0	Base
Vector Control	Message Data	Message Upper Address	Message Address	Entry 1	Base + 1 × 16
Vector Control	Message Data	Message Upper Address	Message Address	Entry 2	Base + 2 × 16
⋮	⋮	⋮	⋮	⋮	⋮
Vector Control	Message Data	Message Upper Address	Message Address	Entry (*N* - 1)	Base + (*N* - 1) × 16

   c. The host calculates the address of the <n^{th}> entry using the following formula:

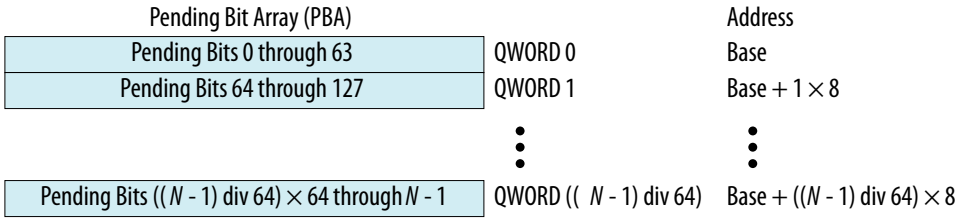
```
nth_address = base address[BAR] + 16<n>
```

2. When Application Layer has an interrupt, it drives an interrupt request to the IRQ Source module.
3. The IRQ Source sets appropriate bit in the MSI-X PBA table.

   The PBA can use qword or dword accesses. For qword accesses, the IRQ Source calculates the address of the <m^{th}> bit using the following formulas:

```
qword address = <PBA base addr> + 8(floor(<m>/64))
qword bit = <m> mod 64
```

Figure 7-7: MSI-X PBA Table


Pending Bit Array (PBA)
Address
Pending Bits 0 through 63
QWORD 0
Base
Pending Bits 64 through 127
QWORD 1
Base + 1 × 8
Pending Bits ((N - 1) div 64) × 64 through N - 1
QWORD ((N - 1) div 64)
Base + ((N - 1) div 64) × 8

4. The IRQ Processor reads the entry in the MSI-X table.
   a. If the interrupt is masked by the `Vector_Control` field of the MSI-X table, the interrupt remains in the pending state.
   b. If the interrupt is not masked, IRQ Processor sends Memory Write Request to the TX slave interface. It uses the address and data from the MSI-X table. If **`Message Upper Address`** = 0, the IRQ Processor creates a three-dword header. If the **`Message Upper Address`** > `0`, it creates a 4-dword header.
5. The host interrupt service routine detects the TLP as an interrupt and services it.

Related Information

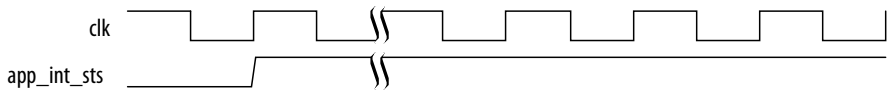
- **Floor and ceiling functions**
- **PCI Local Bus Specification, Rev. 3.0**

Legacy Interrupts

Legacy interrupts are signaled on the PCIe link using message TLPs. The Cyclone V Hard IP for PCI Express generates the message TLPs. The `app_int_sts_vec[7:0]` input vector controls interrupt generation. The assertion of `app_int_sts[<n>]` causes an `Assert_INTA` message TLP to be generated and sent upstream. Deassertion of `app_int_sts[<n>]` causes a `Deassert_INTA` message TLP to be generated and sent upstream. To use legacy interrupts, you must clear the `Interrupt Disable` bit, which is bit 10 of the `Command` register. Then, turn off the `MSI Enable` bit.

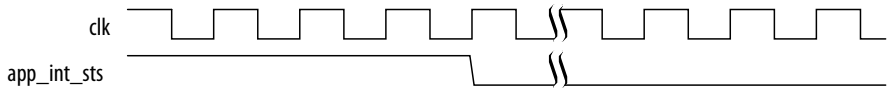
The following figure illustrates interrupt timing for the legacy interface. In this figure the assertion of `app_int_sts` instructs the Hard IP for PCI Express to send a `Assert_INTA` message TLP.

Figure 7-8: Legacy Interrupt Assertion



The following figure illustrates the timing for deassertion of legacy interrupts. The assertion of `app_int_sts` instructs the Hard IP for PCI Express to send a `Deassert_INTA` message.

Figure 7-9: Legacy Interrupt Deassertion



Related Information

Correspondence between Configuration Space Registers and the PCIe Specification on page 5-1

Interrupts for Root Ports

In Root Port mode, the Cyclone V Hard IP for PCI Express receives interrupts through two different mechanisms:

- MSI—Root Ports receive MSI interrupts through the Avalon-ST RX TLP of type `MWr`. This is a memory mapped mechanism.
- Legacy—Legacy interrupts are translated into TLPs of type `Message Interrupt` which is sent to the Application Layer using the `int_status[3:0]` pins.

Normally, the Root Port services rather than sends interrupts; however, in two circumstances the Root Port can send an interrupt to itself to record error conditions:

- When the AER option is enabled, the `aer_msi_num[4:0]` signal indicates which MSI is being sent to the root complex when an error is logged in the AER Capability structure. This mechanism is an alternative to using the `serr_out` signal. The `aer_msi_n um[4:0]` is only used for Root Ports and you must set it to a constant value. It cannot toggle during operation.
- If the Root Port detects a Power Management Event, the `pex_msi_num[4:0]` signal is used by Power Management or Hot Plug to determine the offset between the base message interrupt number and the message interrupt number to send through MSI. The user must set `pex_msi_num[4:0]`to a fixed value.

The `Root Error Status` register reports the status of error messages. The `Root Error Status` register is part of the PCI Express AER Extended Capability structure. It is located at offset 0x830 of the Configuration Space registers.

Error Handling 8

2014.12.15

UG-01110_avst

Subscribe Send Feedback

Each PCI Express compliant device must implement a basic level of error management and can optionally implement advanced error management. The IP core implements both basic and advanced error reporting. Error handling for a Root Port is more complex than that of an Endpoint.

Table 8-1: Error Classification

The *PCI Express Base Specification* defines three types of errors, outlined in the following table.

Type	Responsible Agent	Description
Correctable	Hardware	While correctable errors may affect system performance, data integrity is maintained.
Uncorrectable, non-fatal	Device software	Uncorrectable, non-fatal errors are defined as errors in which data is lost, but system integrity is maintained. For example, the fabric may lose a particular TLP, but it still works without problems.
Uncorrectable, fatal	System software	Errors generated by a loss of data and system failure are considered uncorrectable and fatal. Software must determine how to handle such errors: whether to reset the link or implement other means to minimize the problem.

Related Information

PCI Express Base Specification 2.1 and 3.0

© 2014 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

Physical Layer Errors

Table 8-2: Errors Detected by the Physical Layer

The following table describes errors detected by the Physical Layer. Physical Layer error reporting is optional in the *PCI Express Base Specification*.

Error	Type	Description

Receive port error	Correctable	This error has the following 3 potential causes:
		• Physical coding sublayer error when a lane is in L0 state. These errors are reported to the Hard IP block via the per lane PIPE interface input receive status signals, `rxstatus<lane_number>[2:0]` using the following encodings:
		  • 3'b100: 8B/10B Decode Error
		  • 3'b101: Elastic Buffer Overflow
		  • 3'b110: Elastic Buffer Underflow
		  • 3'b111: Disparity Error
		• Deskew error caused by overflow of the multilane deskew FIFO.
		• Control symbol received in wrong lane.

Data Link Layer Errors

Table 8-3: Errors Detected by the Data Link Layer

Error	Type	Description
Bad TLP	Correctable	This error occurs when a LCRC verification fails or when a sequence number error occurs.
Bad DLLP	Correctable	This error occurs when a CRC verification fails.
Replay timer	Correctable	This error occurs when the replay timer times out.
Replay num rollover	Correctable	This error occurs when the replay number rolls over.
Data Link Layer protocol	Uncorrectable(fatal)	This error occurs when a sequence number specified by the Ack/Nak block in the Data Link Layer (`AckNak_Seq_Num`) does not correspond to an unacknowledged TLP.

Transaction Layer Errors

Table 8-4: Errors Detected by the Transaction Layer

Error	Type	Description
Poisoned TLP received	Uncorrectable (non-fatal)	This error occurs if a received Transaction Layer packet has the EP poison bit set. The received TLP is passed to the Application Layer and the Application Layer logic must take appropriate action in response to the poisoned TLP. Refer to "2.7.2.2 Rules for Use of Data Poisoning" in the *PCI Express Base Specification* for more information about poisoned TLPs.
ECRC check failed (1)	Uncorrectable (non-fatal)	This error is caused by an ECRC check failing despite the fact that the TLP is not malformed and the LCRC check is valid. The Hard IP block handles this TLP automatically. If the TLP is a non-posted request, the Hard IP block generates a completion with completer abort status. In all cases the TLP is deleted in the Hard IP block and not presented to the Application Layer.
Unsupported Request for Endpoints	Uncorrectable (non-fatal)	This error occurs whenever a component receives any of the following Unsupported Requests: • Type 0 Configuration Requests for a non-existing function. • Completion transaction for which the Requester ID does not match the bus, device and function number. • Unsupported message. • A Type 1 Configuration Request TLP for the TLP from the PCIe link. • A locked memory read (MEMRDLK) on native Endpoint. • A locked completion transaction. • A 64-bit memory transaction in which the 32 MSBs of an address are set to 0. • A memory or I/O transaction for which there is no BAR match. • A memory transaction when the Memory Space Enable bit (bit [1] of the PCI Command register at Configuration Space offset 0x4) is set to 0. • A poisoned configuration write request (`CfgWr0`)

Error	Type	Description
		In all cases the TLP is deleted in the Hard IP block and not presented to the Application Layer. If the TLP is a non-posted request, the Hard IP block generates a completion with Unsupported Request status.
Unsupported Requests for Root Port	Uncorrectable (fatal)	This error occurs whenever a component receives an Unsupported Request including: • Unsupported message • A Type 0 Configuration Request TLP • A 64-bit memory transaction which the 32 MSBs of an address are set to 0. • A memory transaction that does not match the address range defined by the Base and Limit Address registers
Completion timeout	Uncorrectable (non-fatal)	This error occurs when a request originating from the Application Layer does not generate a corresponding completion TLP within the established time. It is the responsibility of the Application Layer logic to provide the completion timeout mechanism. The completion timeout should be reported from the Transaction Layer using the `cpl_err[0]` signal.
Completer abort *(1)*	Uncorrectable (non-fatal)	The Application Layer reports this error using the `cpl_err[2]`signal when it aborts receipt of a TLP.

Error	Type	Description
Unexpected completion	Uncorrectable (non-fatal)	This error is caused by an unexpected completion transaction. The Hard IP block handles the following conditions: • The Requester ID in the completion packet does not match the Configured ID of the Endpoint. • The completion packet has an invalid tag number. (Typically, the tag used in the completion packet exceeds the number of tags specified.) • The completion packet has a tag that does not match an outstanding request. • The completion packet for a request that was to I/O or Configuration Space has a length greater than 1 dword. • The completion status is Configuration Retry Status (CRS) in response to a request that was not to Configuration Space. In all of the above cases, the TLP is not presented to the Application Layer; the Hard IP block deletes it. The Application Layer can detect and report other unexpected completion conditions using the `cpl_err[2]` signal. For example, the Application Layer can report cases where the total length of the received successful completions do not match the original read request length.
Receiver overflow *(1)*	Uncorrectable (fatal)	This error occurs when a component receives a TLP that violates the FC credits allocated for this type of TLP. In all cases the hard IP block deletes the TLP and it is not presented to the Application Layer.
Flow control protocol error (FCPE) *(1)*	Uncorrectable (fatal)	This error occurs when a component does not receive update flow control credits with the 200 µs limit.

Error	Type	Description
Malformed TLP	Uncorrectable (fatal)	This error is caused by any of the following conditions: • The data payload of a received TLP exceeds the maximum payload size. • The TD field is asserted but no TLP digest exists, or a TLP digest exists but the TD bit of the PCI Express request header packet is not asserted. • A TLP violates a byte enable rule. The Hard IP block checks for this violation, which is considered optional by the PCI Express specifications. • A TLP in which the type and length fields do not correspond with the total length of the TLP. • A TLP in which the combination of format and type is not specified by the PCI Express specification. • A request specifies an address/length combination that causes a memory space access to exceed a 4 KByte boundary. The Hard IP block checks for this violation, which is considered optional by the PCI Express specification. • Messages, such as Assert_INTX, Power Management, Error Signaling, Unlock, and Set Power Slot Limit, must be transmitted across the default traffic class. The Hard IP block deletes the malformed TLP; it is not presented to the Application Layer.

Note:

1. Considered optional by the *PCI Express Base Specification Revision* .

Error Reporting and Data Poisoning

How the Endpoint handles a particular error depends on the configuration registers of the device.

Refer to the *PCI Express Base Specification 3.0* for a description of the device signaling and logging for an Endpoint.

The Hard IP block implements data poisoning, a mechanism for indicating that the data associated with a transaction is corrupted. Poisoned TLPs have the error/poisoned bit of the header set to 1 and observe the following rules:

- Received poisoned TLPs are sent to the Application Layer and status bits are automatically updated in the Configuration Space.
- Received poisoned Configuration Write TLPs are not written in the Configuration Space.
- The Configuration Space never generates a poisoned TLP; the error/poisoned bit of the header is always set to 0.

Poisoned TLPs can also set the parity error bits in the PCI Configuration Space Status register.

Table 8-5: Parity Error Conditions

Status Bit	Conditions
Detected parity error (status register bit 15)	Set when any received TLP is poisoned.
Master data parity error (status register bit 8)	This bit is set when the command register parity enable bit is set and one of the following conditions is true: • The poisoned bit is set during the transmission of a Write Request TLP. • The poisoned bit is set on a received completion TLP.

Poisoned packets received by the Hard IP block are passed to the Application Layer. Poisoned transmit TLPs are similarly sent to the link.

Related Information

PCI Express Base Specification 2.1 and 3.0

Uncorrectable and Correctable Error Status Bits

The following section is reprinted with the permission of PCI-SIG. Copyright 2010 PCI-SIG.

Figure 8-1: Uncorrectable Error Status Register

The default value of all the bits of this register is 0. An error status bit that is set indicates that the error condition it represents has been detected. Software may clear the error status by writing a 1 to the appropriate bit.

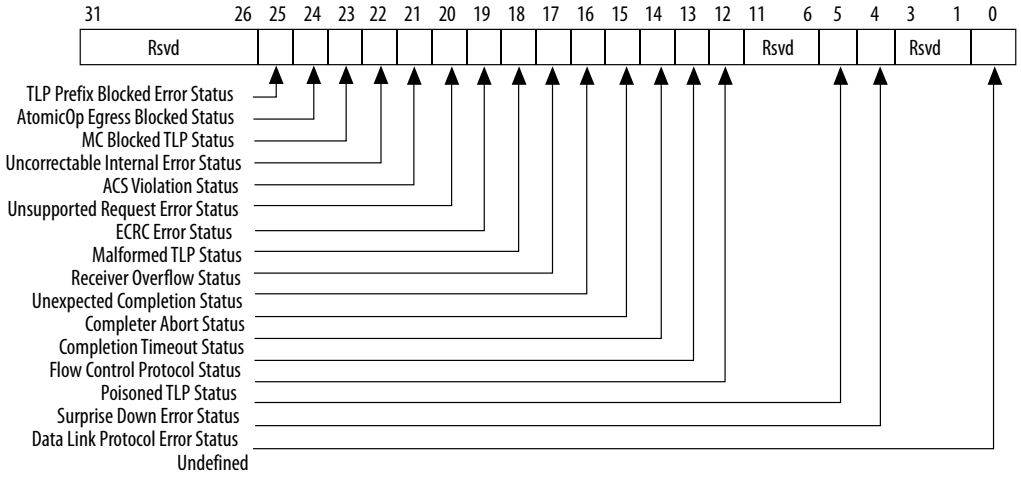
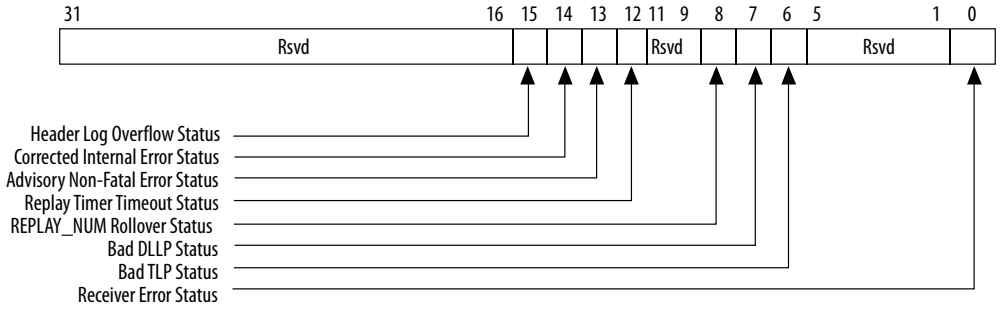


Figure 8-2: Correctable Error Status Register

The default value of all the bits of this register is 0. An error status bit that is set indicates that the error condition it represents has been detected. Software may clear the error status by writing a 1 to the appropriate bit.



IP Core Architecture 9

2014.12.15

UG-01110_avst

Subscribe Send Feedback

The Cyclone V Hard IP for PCI Express implements the complete PCI Express protocol stack as defined in the *PCI Express Base Specification.* The protocol stack includes the following layers:

- *Transaction Layer*—The Transaction Layer contains the Configuration Space, which manages communication with the Application Layer, the RX and TX channels, the RX buffer, and flow control credits.
- *Data Link Layer*—The Data Link Layer, located between the Physical Layer and the Transaction Layer, manages packet transmission and maintains data integrity at the link level. Specifically, the Data Link Layer performs the following tasks:
  - Manages transmission and reception of Data Link Layer Packets (DLLPs)
  - Generates all transmission cyclical redundancy code (CRC) values and checks all CRCs during reception
  - Manages the retry buffer and retry mechanism according to received ACK/NAK Data Link Layer packets
  - Initializes the flow control mechanism for DLLPs and routes flow control credits to and from the Transaction Layer
- *Physical Layer*—The Physical Layer initializes the speed, lane numbering, and lane width of the PCI Express link according to packets received from the link and directives received from higher layers.

© 2014 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

Figure 9-1: Cyclone V Hard IP for PCI Express Using the Avalon-ST Interface

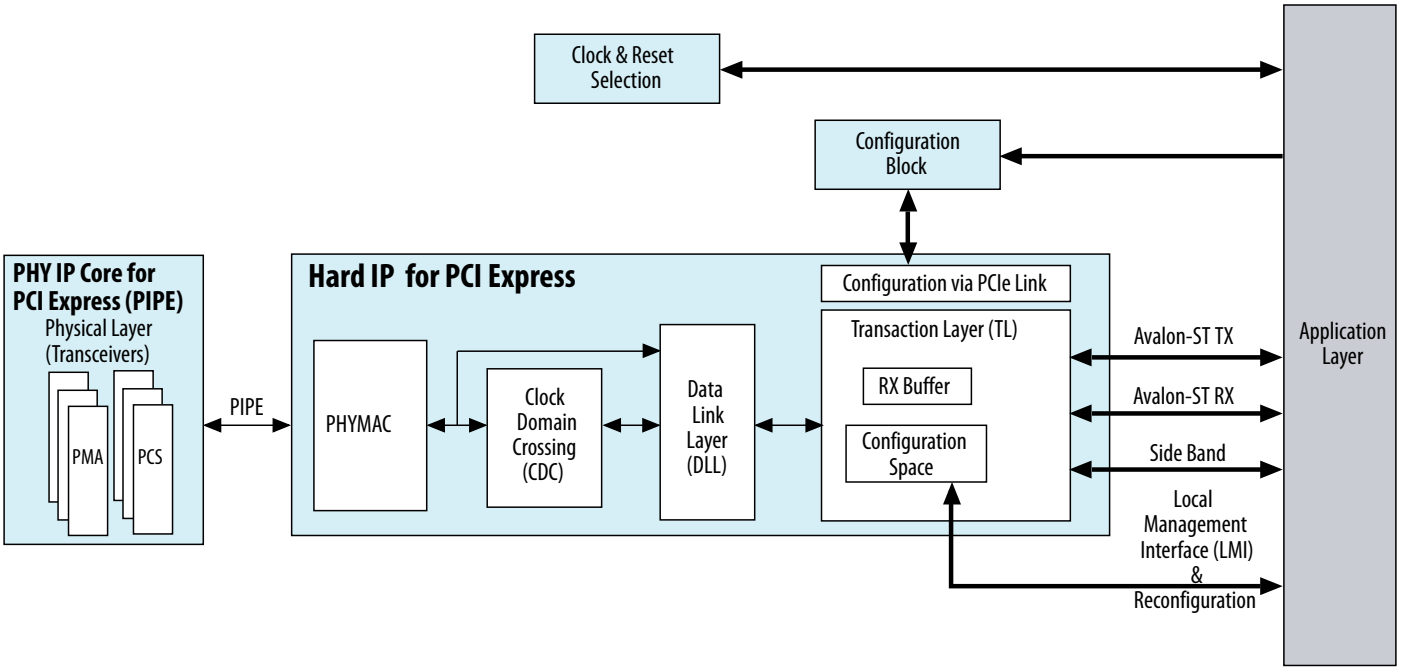


Table 9-1: Application Layer Clock Frequencies

Lanes	Gen1	Gen2
×1	125 MHz @ 64 bits or 62.5 MHz @ 64 bits	125 MHz @ 64 bits
×2	125 MHz @ 64 bits	125 MHz @ 64 bits
×4	125 MHz @ 64 bits	125 MHz @ 128 bits

The following interfaces provide access to the Application Layer's Configuration Space Registers:

- The LMI interface
- The Avalon-MM PCIe reconfiguration interface, which can access any read-only Configuration Space Register
- In Root Port mode, you can also access the Configuration Space Registers with a Configuration TLP using the Avalon-ST interface. A Type 0 Configuration TLP is used to access the Root Port configuration Space Registers, and a Type 1 Configuration TLP is used to access the Configuration Space Registers of downstream components, typically Endpoints on the other side of the link.

The Hard IP includes dedicated clock domain crossing logic (CDC) between the PHYMAC and Data Link Layers.

Related Information

PCI Express Base Specification 2.1 or 3.0

Top-Level Interfaces

Avalon-ST Interface

An Avalon-ST interface connects the Application Layer and the Transaction Layer. This is a point-to-point, streaming interface designed for high throughput applications. The Avalon-ST interface includes the RX and TX datapaths.

For more information about the Avalon-ST interface, including timing diagrams, refer to the *Avalon Interface Specifications*.

RX Datapath

The RX datapath transports data from the Transaction Layer to the Application Layer's Avalon-ST interface. Masking of non-posted requests is partially supported. Refer to the description of the `rx_st_mask` signal for further information about masking.

TX Datapath

The TX datapath transports data from the Application Layer's Avalon-ST interface to the Transaction Layer. The Hard IP provides credit information to the Application Layer for posted headers, posted data, non-posted headers, non-posted data, completion headers and completion data.

The Application Layer may track credits consumed and use the credit limit information to calculate the number of credits available. However, to enforce the PCI Express Flow Control (FC) protocol, the Hard IP also checks the available credits before sending a request to the link, and if the Application Layer violates the available credits for a TLP it transmits, the Hard IP blocks that TLP and all future TLPs until credits become available. By tracking the credit consumed information and calculating the credits available, the Application Layer can optimize performance by selecting for transmission only the TLPs that have credits available.

Related Information

- **Avalon-ST RX Interface** on page 4-2
- **Avalon-ST TX Interface** on page 4-13
- **Avalon Interface Specifications**

Clocks and Reset

The *PCI Express Base Specification* requires an input reference clock, which is called `refclk` in this design. The *PCI Express Base Specification* stipulates that the frequency of this clock be 100 MHz.

The *PCI Express Base Specification* also requires a system configuration time of 100 ms. To meet this specification, IP core includes an embedded hard reset controller. This reset controller exits the reset state after the I/O ring of the device is initialized.

Related Information

- **Clock Signals** on page 4-24

- **Reset, Status, and Link Training Signals** on page 4-24

Local Management Interface (LMI Interface)

The LMI bus provides access to the PCI Express Configuration Space in the Transaction Layer.

Related Information

LMI Signals on page 4-34

Hard IP Reconfiguration

The PCI Express reconfiguration bus allows you to dynamically change the `read-only` values stored in the Configuration Registers.

Related Information

Hard IP Reconfiguration Interface on page 4-43

Transceiver Reconfiguration

The transceiver reconfiguration interface allows you to dynamically reconfigure the values of analog settings in the PMA block of the transceiver. Dynamic reconfiguration is necessary to compensate for process variations.

Related Information

Transceiver PHY IP Reconfiguration on page 15-1

Interrupts

The Hard IP for PCI Express offers the following interrupt mechanisms:

- Message Signaled Interrupts (MSI)— MSI uses the Transaction Layer's request-acknowledge handshaking protocol to implement interrupts. The MSI Capability structure is stored in the Configuration Space and is programmable using Configuration Space accesses.
- MSI-X—The Transaction Layer generates MSI-X messages which are single dword memory writes. In contrast to the MSI capability structure, which contains all of the control and status information for the interrupt vectors, the MSI-X Capability structure points to an MSI-X table structure and MSI-X PBA structure which are stored in memory.
- Legacy interrupts—`app_int_sts_vec[7:0]` controls legacy interrupt generation. When asserted, the Hard IP to generates an `Assert_INT<n>` message TLP.

PIPE

The PIPE interface implements the Intel-designed PIPE interface specification. You can use this parallel interface to speed simulation; however, you cannot use the PIPE interface in actual hardware. The Gen1 and Gen2 simulation models support PIPE and serial simulation.

Transaction Layer

The Transaction Layer is located between the Application Layer and the Data Link Layer. It generates and receives Transaction Layer Packets. The following illustrates the Transaction Layer. The Transaction Layer includes three sub-blocks: the TX datapath, Configuration Space, and RX datapath.

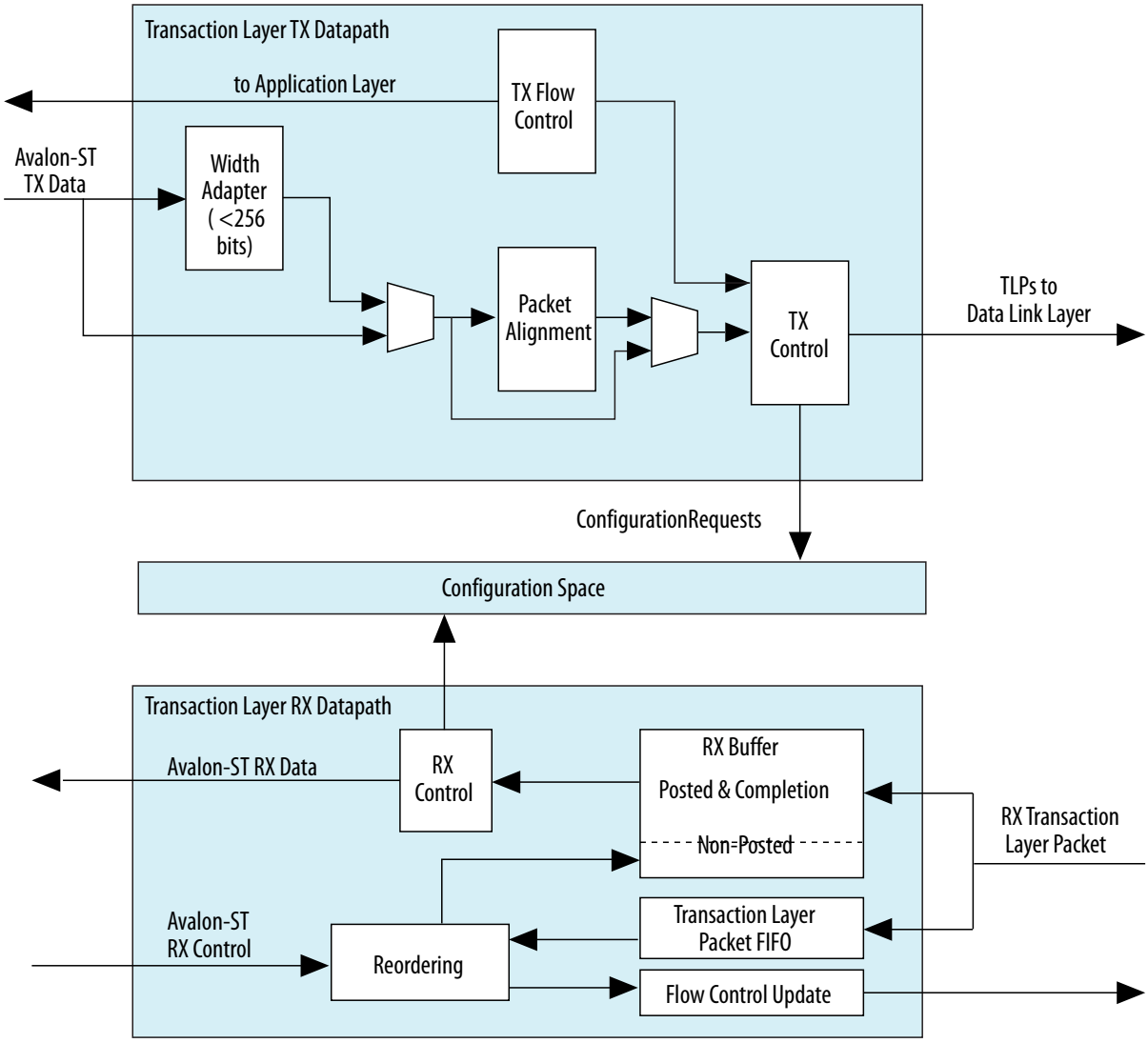
Tracing a transaction through the RX datapath includes the following steps:

1. The Transaction Layer receives a TLP from the Data Link Layer.
2. The Configuration Space determines whether the TLP is well formed and directs the packet based on traffic class (TC).
3. TLPs are stored in a specific part of the RX buffer depending on the type of transaction (posted, non-posted, and completion).
4. The TLP FIFO block stores the address of the buffered TLP.
5. The receive reordering block reorders the queue of TLPs as needed, fetches the address of the highest priority TLP from the TLP FIFO block, and initiates the transfer of the TLP to the Application Layer.
6. When ECRC generation and forwarding are enabled, the Transaction Layer forwards the ECRC dword to the Application Layer.

Tracing a transaction through the TX datapath involves the following steps:

1. The Transaction Layer informs the Application Layer that sufficient flow control credits exist for a particular type of transaction using the TX credit signals. The Application Layer may choose to ignore this information.
2. The Application Layer requests permission to transmit a TLP. The Application Layer must provide the transaction and must be prepared to provide the entire data payload in consecutive cycles.
3. The Transaction Layer verifies that sufficient flow control credits exist and acknowledges or postpones the request.
4. The Transaction Layer forwards the TLP to the Data Link Layer.

Figure 9-2: Architecture of the Transaction Layer: Dedicated Receive Buffer



Configuration Space

The Configuration Space implements the following configuration registers and associated functions:

- Header Type 0 Configuration Space for Endpoints
- Header Type 1 Configuration Space for Root Ports
- PCI Power Management Capability Structure
- Virtual Channel Capability Structure
- Message Signaled Interrupt (MSI) Capability Structure
- Message Signaled Interrupt–X (MSI–X) Capability Structure
- PCI Express Capability Structure
- Advanced Error Reporting (AER) Capability Structure
- Vendor Specific Extended Capability (VSEC)

The Configuration Space also generates all messages (PME#, INT, error, slot power limit), MSI requests, and completion packets from configuration requests that flow in the direction of the root complex, except slot power limit messages, which are generated by a downstream port. All such transactions are dependent upon the content of the PCI Express Configuration Space as described in the *PCI Express Base Specification*.

Related Information

- **Type 0 Configuration Space Registers** on page 5-5
- **Type 1 Configuration Space Registers**
- **PCI Express Base Specification Revision 2.1 or 3.0**

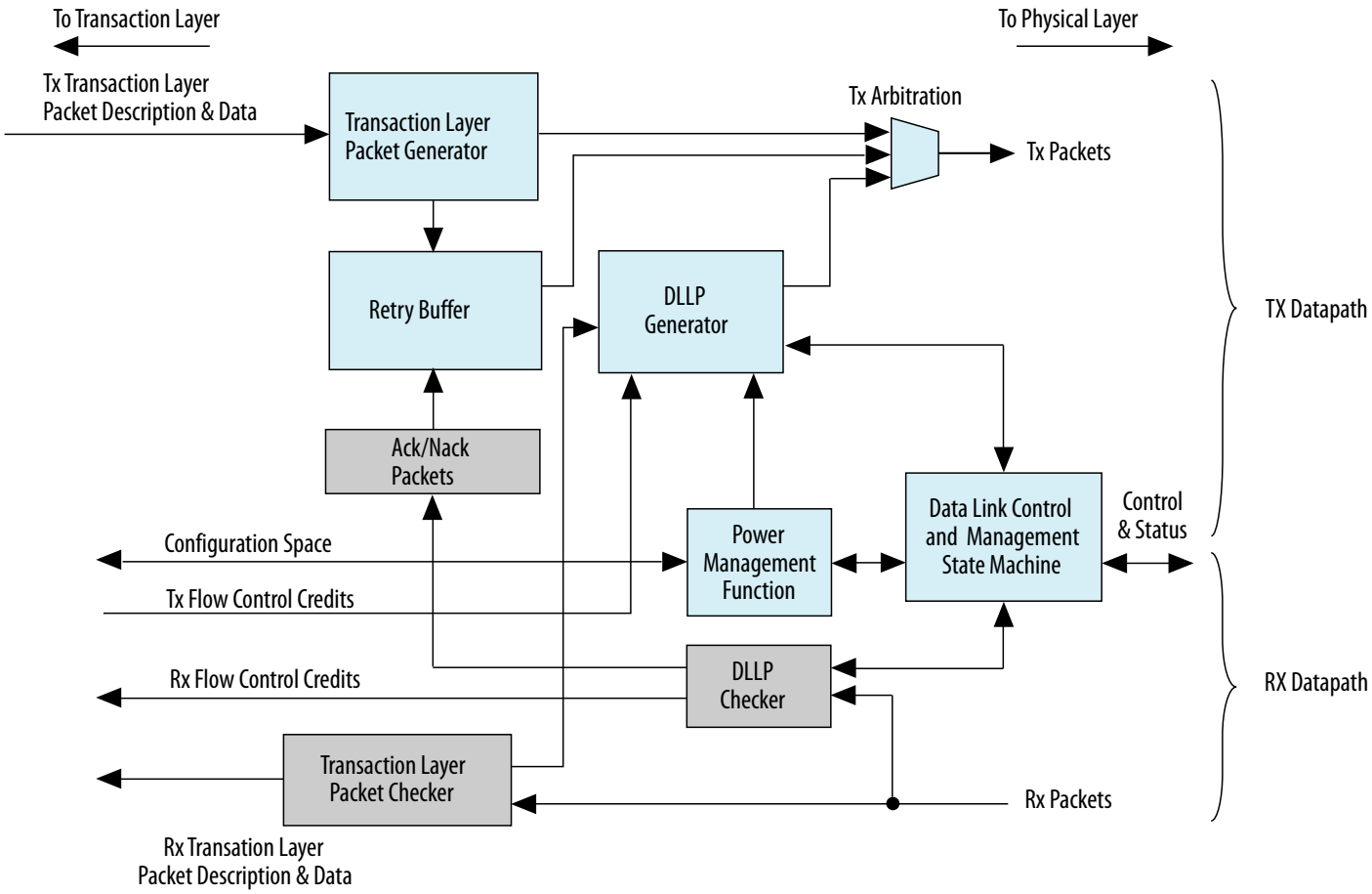
Data Link Layer

The Data Link Layer is located between the Transaction Layer and the Physical Layer. It maintains packet integrity and communicates (by DLL packet transmission) at the PCI Express link level (as opposed to component communication by TLP transmission in the interconnect fabric).

The DLL implements the following functions:

- Link management through the reception and transmission of DLL packets (DLLP), which are used for the following functions:
  - Power management of DLLP reception and transmission
  - To transmit and receive ACK/NACK packets
  - Data integrity through generation and checking of CRCs for TLPs and DLLPs
  - TLP retransmission in case of NAK DLLP reception using the retry buffer
  - Management of the retry buffer
  - Link retraining requests in case of error through the Link Training and Status State Machine (LTSSM) of the Physical Layer

Figure 9-3: Data Link Layer



The DLL has the following sub-blocks:

- Data Link Control and Management State Machine—This state machine is synchronized with the Physical Layer's LTSSM state machine and is also connected to the Configuration Space Registers. It initializes the link and flow control credits and reports status to the Configuration Space.
- Power Management—This function handles the handshake to enter low power mode. Such a transition is based on register values in the Configuration Space and received Power Management (PM) DLLPs.
- Data Link Layer Packet Generator and Checker—This block is associated with the DLLP's 16-bit CRC and maintains the integrity of transmitted packets.
- Transaction Layer Packet Generator—This block generates transmit packets, generating a sequence number and a 32-bit CRC (LCRC). The packets are also sent to the retry buffer for internal storage. In retry mode, the TLP generator receives the packets from the retry buffer and generates the CRC for the transmit packet.
- Retry Buffer—The retry buffer stores TLPs and retransmits all unacknowledged packets in the case of NAK DLLP reception. In case of ACK DLLP reception, the retry buffer discards all acknowledged packets.

- ACK/NAK Packets—The ACK/NAK block handles ACK/NAK DLLPs and generates the sequence number of transmitted packets.
- Transaction Layer Packet Checker—This block checks the integrity of the received TLP and generates a request for transmission of an ACK/NAK DLLP.
- TX Arbitration—This block arbitrates transactions, prioritizing in the following order:
  - Initialize FC Data Link Layer packet
  - ACK/NAK DLLP (high priority)
  - Update FC DLLP (high priority)
  - PM DLLP
  - Retry buffer TLP
  - TLP
  - Update FC DLLP (low priority)
  - ACK/NAK FC DLLP (low priority)

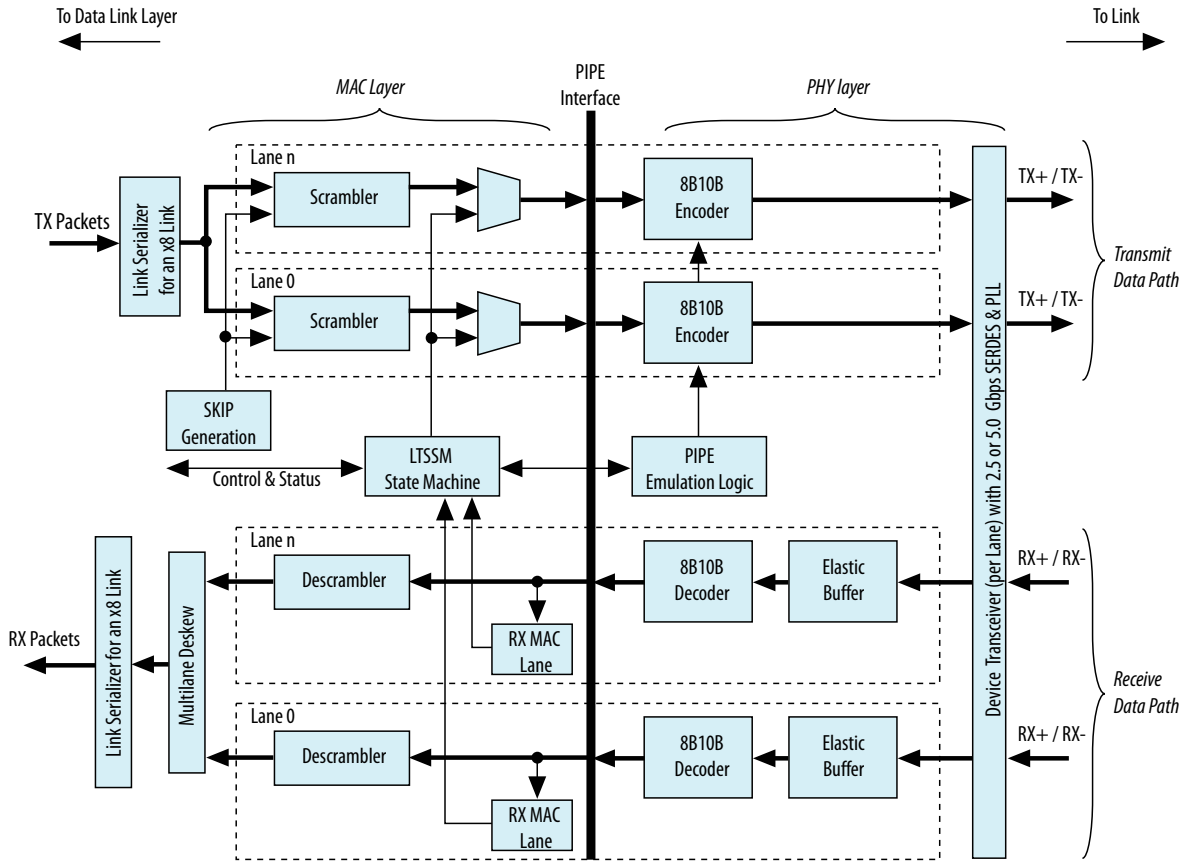
Physical Layer

The Physical Layer is the lowest level of the PCI Express protocol stack. It is the layer closest to the serial link. It encodes and transmits packets across a link and accepts and decodes received packets. The Physical Layer connects to the link through a high-speed SERDES interface running at 2.5 Gbps for Gen1 implementations and a5 2.5 or 5.0 Gbps for Gen2 implementations.

The Physical Layer is responsible for the following actions:

- Initializing the link
- Scrambling/descrambling and 8B/10B encoding/decoding for 2.5 Gbps (Gen1) and 5.0 Gbps (Gen2) per lane
- Serializing and deserializing data
- Operating the PIPE 3.0 Interface
- Implementing auto speed negotiation (Gen2 and Gen3)
- Transmitting and decoding the training sequence
- Providing hardware autonomous speed control
- Implementing auto lane reversal

Figure 9-4: Physical Layer Architecture



The Physical Layer is subdivided by the PIPE Interface Specification into two layers (bracketed horizontally in above figure):

- Media Access Controller (MAC) Layer—The MAC layer includes the LTSSM and the scrambling/descrambling and multilane deskew functions.
- PHY Layer—The PHY layer includes the 8B/10B and 128b/130b encode/decode functions, elastic buffering, and serialization/deserialization functions.

The Physical Layer integrates both digital and analog elements. Intel designed the PIPE interface to separate the MAC from the PHY. The Cyclone V Hard IP for PCI Express complies with the PIPE interface specification.

The PHYMAC block comprises four main sub-blocks:

- MAC Lane—Both the RX and the TX path use this block.
  - On the RX side, the block decodes the Physical Layer packet and reports to the LTSSM the type and number of TS1/TS2 ordered sets received.
  - On the TX side, the block multiplexes data from the DLL and the LTSTX sub-block. It also adds lane specific information, including the lane number and the force PAD value when the LTSSM disables the lane during initialization.
- LTSSM—This block implements the LTSSM and logic that tracks TX and RX data on each lane.
- For transmission, it interacts with each MAC lane sub-block and with the LTSTX sub-block by asserting both global and per-lane control bits to generate specific Physical Layer packets.
  - On the receive path, it receives the Physical Layer packets reported by each MAC lane sub-block. It also enables the multilane deskew block. This block reports the Physical Layer status to higher layers.
  - LTSTX (Ordered Set and SKP Generation)—This sub-block generates the Physical Layer packet. It receives control signals from the LTSSM block and generates Physical Layer packet for each lane. It generates the same Physical Layer Packet for all lanes and PAD symbols for the link or lane number in the corresponding TS1/TS2 fields.

The block also handles the receiver detection operation to the PCS sub-layer by asserting predefined PIPE signals and waiting for the result. It also generates a SKP Ordered Set at every predefined timeslot and interacts with the TX alignment block to prevent the insertion of a SKP Ordered Set in the middle of packet.

- Deskew—This sub-block performs the multilane deskew function and the RX alignment between the number of initialized lanes and the 64-bit data path.

The multilane deskew implements an eight-word FIFO buffer for each lane to store symbols. Each symbol includes eight data bits, one disparity bit, and one control bit. The FIFO discards the FTS, COM, and SKP symbols and replaces PAD and IDL with D0.0 data. When all eight FIFOs contain data, a read can occur.

When the multilane lane deskew block is first enabled, each FIFO begins writing after the first COM is detected. If all lanes have not detected a COM symbol after seven clock cycles, they are reset and the resynchronization process restarts, or else the RX alignment function recreates a 64-bit data word which is sent to the DLL.

Multi-Function Support

The Cyclone V Hard IP for PCI Express supports up to eight functions for Endpoints. You set up the each function under the Port Functions heading in the parameter editor. You can configure Cyclone V devices to include both Native and Legacy Endpoints. Each function replicates the Configuration Space Registers, including logic for Tag Tracking and Error detection.

Because the Configuration Space is replicated for each function, some Configuration Space Register settings may conflict. Arbitration logic resolves differences when settings contain different values across multiple functions. The arbitration logic implements the rules for resolving conflicts as specified in the

PCI Express Base Specification 2.1. Examples of settings that require arbitration include the following features:

- Link Control settings
- Error detection and logging for non-function-specific errors
- Error message collapsing
- Maximum payload size (All functions use the largest specified maximum payload setting.)

  Note: Altera strongly recommends that your software configure the **Maximum payload size** (in the `Device Control` register) with the same value across all functions.
- Interrupt message collapsing

You can access the Configuration Space Registers for the active function using the LMI interface. In Root Port mode, you can also access the Configuration Space Registers using a Configuration Type TLP. Refer to *Type 0 Configuration Space Registers* for more information about the Configuration Space Registers.

Transaction Layer Protocol (TLP) Details 10

2014.12.15

UG-01110_avst

Supported Message Types

INTX Messages

The following table describes the message INTX messages. For Endpoints, only INTA messages are generated.

Table 10-1: INTX Messages

Message	Root Port	Endpoint	Generated by			Comments
			App Layer	Core	Core (with App Layer input)	
INTX Mechanism						For Endpoints, only INTA messages are generated.
Assert_INTA	Receive	Transmit	No	Yes	No	For Root Port, legacy interrupts are translated into message interrupt TLPs which triggers the `int_status[3:0]` signals to the Application Layer. • `int_status[0]`: Interrupt signal A • `int_status[1]`: Interrupt signal B • `int_status[2]`: Interrupt signal C • `int_status[3]`: Interrupt signal D
Assert_INTB	Receive	Transmit	No	No	No	
Assert_INTC	Receive	Transmit	No	No	No	
Assert_INTD	Receive	Transmit	No	No	No	
Deassert_INTA	Receive	Transmit	No	Yes	No	

© 2014 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

Message	Root Port	Endpoint	Generated by			Comments
			App Layer	Core	Core (with App Layer input)	
Deassert_INTB	Receive	Transmit	No	No	No	
Deassert_INTC	Receive	Transmit	No	No	No	
Deassert_INTD	Receive	Transmit	No	No	No	

Power Management Messages

Table 10-2: Power Management Messages

Message	Root Port	Endpoint	Generated by			Comments
			App Layer	Core	Core (with App Layer input)	
PM_Active_State_Nak	TX	RX	No	Yes	No	—
PM_PME	RX	TX	No	No	Yes	—
PME_Turn_Off	TX	RX	No	No	Yes	The `pme_to_cr` signal sends and acknowledges this message: • Root Port: When `pme_to_cr` is asserted, the Root Port sends the `PME_turn_off` message. • Endpoint: When `PME_to_cr` is asserted, the Endpoint acknowledges the `PME_turn_off` message by sending a `pme_to_ack` message to the Root Port.
PME_TO_Ack	RX	TX	No	No	Yes	—

Error Signaling Messages

Table 10-3: Error Signaling Messages

Message	Root Port	Endpoint	Generated by			Comments
			App Layer	Core	Core (with App Layer input)	
ERR_COR	RX	TX	No	Yes	No	In addition to detecting errors, a Root Port also gathers and manages errors sent by downstream components through the ERR_COR, ERR_NONFATAL, AND ERR_FATAL Error Messages. In Root Port mode, there are two mechanisms to report an error event to the Application Layer: • `serr_out` output signal. When set, indicates to the Application Layer that an error has been logged in the AER capability structure • `aer_msi_num` input signal. When the **Implement advanced error reporting** option is turned on, you can set `aer_msi_num` to indicate which MSI is being sent to the root complex when an error is logged in the AER Capability structure.
ERR_NONFATAL	RX	TX	No	Yes	No	—
ERR_FATAL	RX	TX	No	Yes	No	—

Locked Transaction Message

Table 10-4: Locked Transaction Message

Message	Root Port	Endpoint	Generated by			Comments
			App Layer	Core	Core (with App Layer input)	
Unlock Message	Transmit	Receive	Yes	No	No	

Slot Power Limit Message

The *PCI Express Base Specification Revision* states that this message is not mandatory after link training.

Table 10-5: Slot Power Message

Message	Root Port	Endpoint	Generated by			Comments
			App Layer	Core	Core (with App Layer input)	
Set Slot Power Limit	Transmit	Receive	No	Yes	No	In Root Port mode, through software.

Related Information

PCI Express Base Specification Revision 2.1 or 3.0

Vendor-Defined Messages

Table 10-6: Vendor-Defined Message

Message	Root Port	Endpoint	Generated by			Comments
			App Layer	Core	Core (with App Layer input)	
Vendor Defined Type 0	Transmit Receive	Transmit Receive	Yes	No	No	

Message	Root Port	Endpoint	Generated by			Comments
			App Layer	Core	Core (with App Layer input)	
Vendor Defined Type 1	Transmit Receive	Transmit Receive	Yes	No	No	

Hot Plug Messages

Table 10-7: Locked Transaction Message

Message	Root Port	Endpoint	Generated by			Comments
			App Layer	Core	Core (with App Layer input)	
Attention_ indicator On	Transmit	Receive	No	Yes	No	Per the recommendations in the *PCI Express Base Specification Revision* , these messages are not transmitted to the Application Layer.
Attention_ Indicator Blink	Transmit	Receive	No	Yes	No	
Attention_ indicator Off	Transmit	Receive	No	Yes	No	
Power_ Indicator On	Transmit	Receive	No	Yes	No	
Power_ Indicator Blink	Transmit	Receive	No	Yes	No	
Power_ Indicator Off	Transmit	Receive	No	Yes	No	

Message	Root Port	Endpoint	Generated by			Comments
			App Layer	Core	Core (with App Layer input)	
Attention Button_Pressed (Endpoint only)	Receive	Transmit	No	No	Yes	N/A

Related Information

PCI Express Base Specification Revision 2.1 or 3.0

Transaction Layer Routing Rules

Transactions adhere to the following routing rules:

- In the receive direction (from the PCI Express link), memory and I/O requests that match the defined base address register (BAR) contents and vendor-defined messages with or without data route to the receive interface. The Application Layer logic processes the requests and generates the read completions, if needed.
- In Endpoint mode, received Type 0 Configuration requests from the PCI Express upstream port route to the internal Configuration Space and the Cyclone V Hard IP for PCI Express generates and transmits the completion.
- The Hard IP handles supported received message transactions (Power Management and Slot Power Limit) internally. The Endpoint also supports the Unlock and Type 1 Messages. The Root Port supports Interrupt, Type 1, and error Messages.
- Vendor-defined Type 0 Message TLPs are passed to the Application Layer.
- The Transaction Layer treats all other received transactions (including memory or I/O requests that do not match a defined BAR) as Unsupported Requests. The Transaction Layer sets the appropriate error bits and transmits a completion, if needed. These Unsupported Requests are not made visible to the Application Layer; the header and data are dropped.
- For memory read and write request with addresses below 4 GB, requestors must use the 32-bit format. The Transaction Layer interprets requests using the 64-bit format for addresses below 4 GB as an Unsupported Request and does not send them to the Application Layer. If Error Messaging is enabled, an error Message TLP is sent to the Root Port. Refer to *Transaction Layer Errors* for a comprehensive list of TLPs the Hard IP does not forward to the Application Layer.
- The Transaction Layer sends all memory and I/O requests, as well as completions generated by the Application Layer and passed to the transmit interface, to the PCI Express link.
- The Hard IP can generate and transmit power management, interrupt, and error signaling messages automatically under the control of dedicated signals. Additionally, it can generate MSI requests under the control of the dedicated signals.
- In Root Port mode, the Application Layer can issue Type 0 or Type 1 Configuration TLPs on the Avalon-ST TX bus.

- The Type 0 Configuration TLPs are only routed to the Configuration Space of the Hard IP and are not sent downstream on the PCI Express link.
- The Type 1 Configuration TLPs are sent downstream on the PCI Express link. If the bus number of the Type 1 Configuration TLP matches the Secondary Bus Number register value in the Root Port Configuration Space, the TLP is converted to a Type 0 TLP.
- For more information about routing rules in Root Port mode, refer to *Section 7.3.3 Configuration Request Routing Rules* in the *PCI Express Base Specification* .

Related Information

- **Transaction Layer Errors** on page 8-3
- **PCI Express Base Specification Revision 2.1 or 3.0**

Receive Buffer Reordering

The PCI, PCI-X and PCI Express protocols include ordering rules for concurrent TLPs. Ordering rules are necessary for the following reasons:

- To guarantee that TLPs complete in the intended order
- To avoid deadlock
- To maintain computability with ordering used on legacy buses
- To maximize performance and throughput by minimizing read latencies and managing read/write ordering
- To avoid race conditions in systems that include legacy PCI buses by guaranteeing that reads to an address do not complete before an earlier write to the same address

PCI uses a strongly-ordered model with some exceptions to avoid potential deadlock conditions. PCI-X added a relaxed ordering (RO) bit in the TLP header. It is bit 5 of byte 2 in the TLP header, or the high-order bit of the `attributes` field in the TLP formats shown in Chapter A, Transaction Layer Packet (TLP) Header Formats. If this bit is set, relaxed ordering is permitted. If software can guarantee that no dependencies exist between pending transactions, you can safely set the relaxed ordering bit.

The following table summarizes the ordering rules from the PCI specification. In this table, the entries have the following meanings:

- Columns represent the first transaction issued.
- Rows represent the next transaction.
- At each intersection, the implicit question is: should this row packet be allowed to pass the column packet? The following three answers are possible:
  - Yes: the second transaction must be allowed to pass the first to avoid deadlock.
  - Y/N: There are no requirements. A device may allow the second transaction to pass the first.
  - No: The second transaction must not be allowed to pass the first.

The following transaction ordering rules apply to the table below.

- A Memory Write or Message Request with the Relaxed Ordering Attribute bit clear (b'0) must not pass any other Memory Write or Message Request.
- A Memory Write or Message Request with the Relaxed Ordering Attribute bit set (b'1) is permitted to pass any other Memory Write or Message Request.
- Endpoints, Switches, and Root Complex may allow Memory Write and Message Requests to pass Completions or be blocked by Completions.
- Memory Write and Message Requests can pass Completions traveling in the PCI Express to PCI directions to avoid deadlock.
- If the Relaxed Ordering attribute is not set, then a Read Completion cannot pass a previously enqueued Memory Write or Message Request.
- If the Relaxed Ordering attribute is set, then a Read Completion is permitted to pass a previously enqueued Memory Write or Message Request.
- Read Completion associated with different Read Requests are allowed to be blocked by or to pass each other.
- Read Completions for Request (same Transaction ID) must return in address order.
- Non-posted requests cannot pass other non-posted requests.
- `CfgRd0`CfgRd0 can pass `IORd` or `MRd`.
- `CfgWr0` can `IORd` or `MRd`.
- `CfgRd0` can pass `IORd` or `MRd`.
- `CfrWr0` can pass `IOWr`.

Table 10-8: Transaction Ordering Rules

Can the Row Pass the Column?		Posted Req		Non Posted Req				Completion	
		Memory Write or Message Req		Read Request		I/O or Cfg Write Req			
		Spec	Hard IP	Spec	Hard IP	Spec	Hard IP	Spec	Hard IP
P	Posted Req	No Y/N	No No	Yes	Yes	Yes	Yes	Y/N Yes	No No
NP	Read Req	No	No	Y/N	No	Y/N	No	Y/N	No
	Non-Posted Req with data	No	No	Y/N	No	Y/N	No	Y/N	No

Can the Row Pass the Column?		Posted Req		Non Posted Req				Completion	
		Memory Write or Message Req		Read Request		I/O or Cfg Write Req			
Cmpl	Cmpl	No Y/N	No No	Yes	Yes	Yes	Yes	Y/N No	No No
	I/O or Configuration Write Cmpl	Y/N	No	Yes	Yes	Yes	Yes	Y/N	No

As the table above indicates, the RX datapath implements an RX buffer reordering function that allows Posted and Completion transactions to pass Non-Posted transactions (as allowed by PCI Express ordering rules) when the Application Layer is unable to accept additional Non-Posted transactions.

The Application Layer dynamically enables the RX buffer reordering by asserting the `rx_mask` signal. The `rx_mask` signal blocks non-posted Req transactions made to the Application Layer interface so that only posted and completion transactions are presented to the Application Layer.

Note: MSI requests are conveyed in exactly the same manner as PCI Express memory write requests and are indistinguishable from them in terms of flow control, ordering, and data integrity.

Related Information

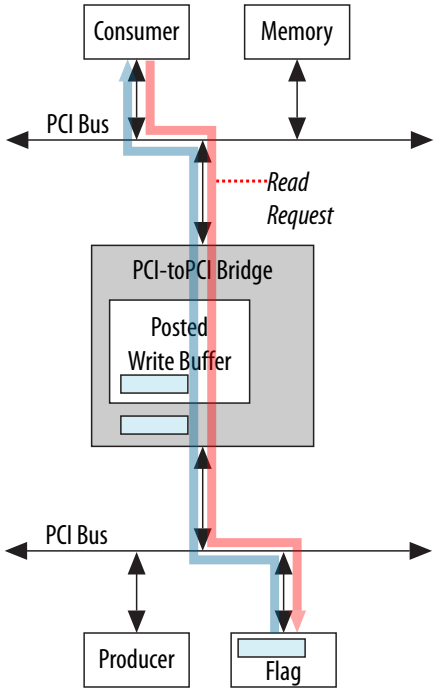
PCI Express Base Specification Revision 2.1 or 3.0

Using Relaxed Ordering

Transactions from unrelated threads are unlikely to have data dependencies. Consequently, you may be able to use relaxed ordering to improve system performance. The drawback is that only some transactions can be optimized for performance. Complete the following steps to decide whether to enable relaxed ordering in your design:

1. Create a system diagram showing all PCI Express and legacy devices.
2. Analyze the relationships between the components in your design to identify the following hazards:
   a. Race conditions: A race condition exists if a read to a location can occur before a previous write to that location completes. The following figure shows a data producer and data consumer on opposite sides of a PCI-to-PCI bridge. The producer writes data to the memory through a PCI-to-PCI bridge. The consumer must read a flag to confirm the producer has written the new data into the memory before reading the data. However, because the PCI-to-PCI bridge includes a write buffer, the flag may indicate that it is safe to read data while the actual data remains in the PCI-to-PCI bridge posted write buffer.

Figure 10-1: Design Including Legacy PCI Buses Requiring Strong Ordering

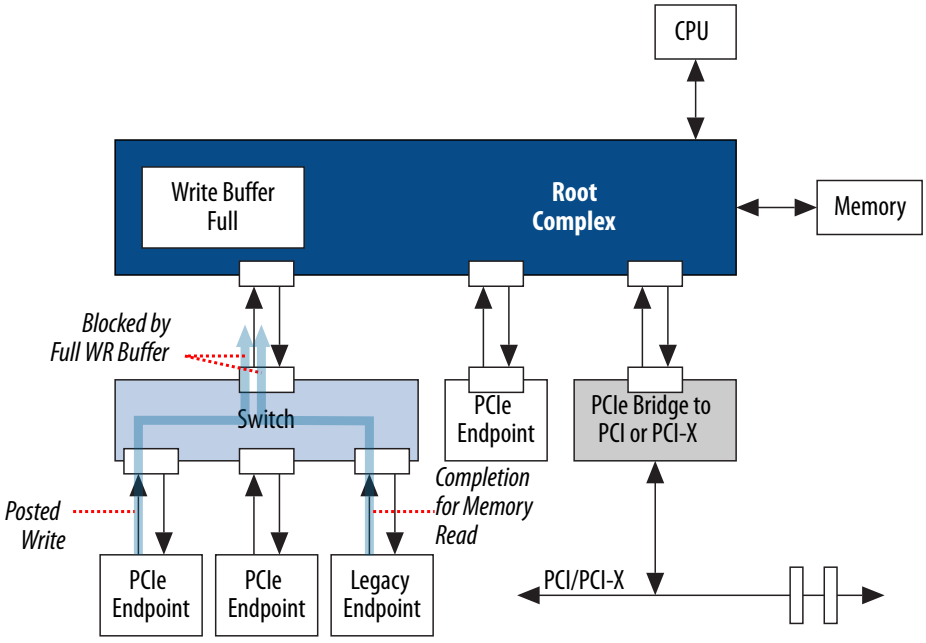


b. A shared memory architecture where more than one thread accesses the same locations in memory.

If either of these conditions exists, relaxed ordering will lead to incorrect results.

3. If your analysis determines that relaxed ordering does not lead to possible race conditions or read or write hazards, you can enable relaxed ordering by setting the RO bit in the TLP header.
4. The following figure shows two PCIe Endpoints and Legacy Endpoint connected to a switch. The three PCIe Endpoints are not likely to have data dependencies. Consequently, it would be safe to set the relaxed ordering bit for devices connected to the switch. In this system, if relax ordering is not enabled, a memory read to the legacy Endpoint is blocked. The legacy Endpoint read is blocked because an earlier posted write cannot be completed as the write buffer is full. .

Figure 10-2: PCI Express Design Using Relaxed Ordering



5. If your analysis indicates that you can enable relaxed ordering, simulate your system with and without relaxed ordering enabled. Compare the results and performance.
6. If relaxed ordering improves performance without introducing errors, you can enable it in your system.

Throughput Optimization 11

2014.12.15

UG-01110_avst

Subscribe Send Feedback

The *PCI Express Base Specification* defines a flow control mechanism to ensure efficient transfer of TLPs.

Each transmitter, the write requester in this case, maintains a `credit limit` register and a `credits consumed` register. The `credit limit` register is the sum of all credits received by the receiver, the write completer in this case. The `credit limit` register is initialized during the flow control initialization phase of link initialization and then updated during operation by Flow Control (FC) Update DLLPs. The `credits consumed` register is the sum of all credits consumed by packets transmitted. Separate `credit limit` and `credits consumed` registers exist for each of the six types of Flow Control:

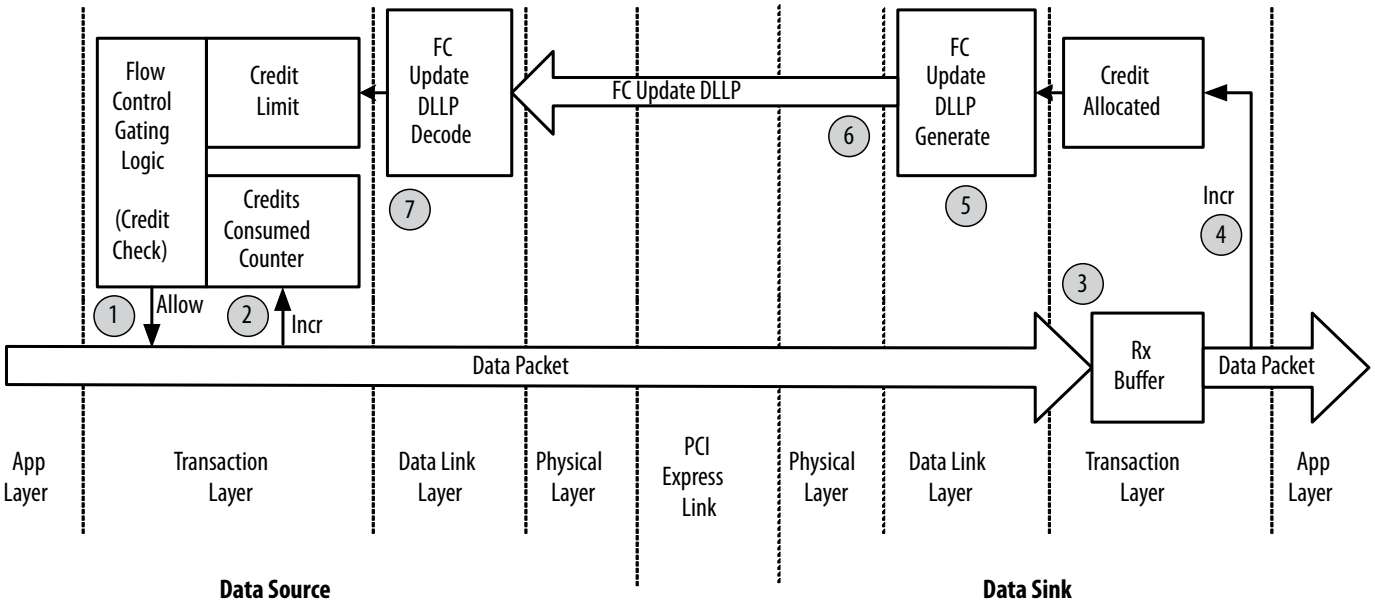
- Posted Headers
- Posted Data
- Non-Posted Headers
- Non-Posted Data
- Completion Headers
- Completion Data

Each receiver also maintains a `credit allocated` counter which is initialized to the total available space in the RX buffer (for the specific Flow Control class) and then incremented as packets are pulled out of the RX buffer by the Application Layer. The value of this register is sent as the FC Update DLLP value.

© 2014 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

Figure 11-1: Flow Control Update Loop



The following numbered steps describe each step in the Flow Control Update loop. The corresponding numbers in the figure show the general area to which they correspond.

1. When the Application Layer has a packet to transmit, the number of credits required is calculated. If the current value of the credit limit minus credits consumed is greater than or equal to the required credits, then the packet can be transmitted immediately. However, if the credit limit minus credits consumed is less than the required credits, then the packet must be held until the credit limit is increased to a sufficient value by an FC Update DLLP. This check is performed separately for the header and data credits; a single packet consumes only a single header credit.
2. After the packet is selected for transmission the `credits consumed` register is incremented by the number of credits consumed by this packet. This increment happens for both the header and data `credit consumed` registers.
3. The packet is received at the other end of the link and placed in the RX buffer.
4. At some point the packet is read out of the RX buffer by the Application Layer. After the entire packet is read out of the RX buffer, the `credit allocated` register can be incremented by the number of credits the packet has used. There are separate `credit allocated` registers for the header and data credits.
5. The value in the `credit allocated` register is used to create an FC Update DLLP.
6. After an FC Update DLLP is created, it arbitrates for access to the PCI Express link. The FC Update DLLPs are typically scheduled with a low priority; consequently, a continuous stream of Application Layer TLPs or other DLLPs (such as ACKs) can delay the FC Update DLLP for a long time. To prevent starving the attached transmitter, FC Update DLLPs are raised to a high priority under the following three circumstances:
   a. When the last sent `credit allocated` counter minus the amount of received data is less than `MAX_PAYLOAD` and the current `credit allocated` counter is greater than the last sent credit

counter. Essentially, this means the data sink knows the data source has less than a full `MAX_PAYLOAD` worth of credits, and therefore is starving.

b. When an internal timer expires from the time the last FC Update DLLP was sent, which is configured to 30 µs to meet the *PCI Express Base Specification* for resending FC Update DLLPs.

c. When the `credit allocated` counter minus the last sent `credit allocated` counter is greater than or equal to 25% of the total credits available in the RX buffer, then the FC Update DLLP request is raised to high priority.

After arbitrating, the FC Update DLLP that won the arbitration to be the next item is transmitted. In the worst case, the FC Update DLLP may need to wait for a maximum sized TLP that is currently being transmitted to complete before it can be sent.

7. The original write requester receives the FC Update DLLP. The `credit limit` value is updated. If packets are stalled waiting for credits, they can now be transmitted.

Note: You must keep track of the credits consumed by the Application Layer.

Throughput of Posted Writes

The throughput of posted writes is limited primarily by the Flow Control Update loop as shown in **Figure 11-1**. If the write requester sources the data as quickly as possible, and the completer consumes the data as quickly as possible, then the Flow Control Update loop may be the biggest determining factor in write throughput, after the actual bandwidth of the link.

The figure below shows the main components of the Flow Control Update loop with two communicating PCI Express ports:

- Write Requester
- Write Completer

To allow the write requester to transmit packets continuously, the `credit allocated` and the `credit limit` counters must be initialized with sufficient credits to allow multiple TLPs to be transmitted while waiting for the FC Update DLLP that corresponds to the freeing of credits from the very first TLP transmitted.

You can use the **RX Buffer space allocation - Desired performance for received requests** to configure the RX buffer with enough space to meet the credit requirements of your system.

Related Information

PCI Express Base Specification 2.1 or 3.0

Throughput of Non-Posted Reads

To support a high throughput for read data, you must analyze the overall delay from the time the Application Layer issues the read request until all of the completion data is returned. The Application Layer must be able to issue enough read requests, and the read completer must be capable of processing these read requests quickly enough (or at least offering enough non-posted header credits) to cover this delay.

However, much of the delay encountered in this loop is well outside the IP core and is very difficult to estimate. PCI Express switches can be inserted in this loop, which makes determining a bound on the delay more difficult.

Nevertheless, maintaining maximum throughput of completion data packets is important. Endpoints must offer an infinite number of completion credits. Endpoints must buffer this data in the RX buffer until the Application Layer can process it. Because the Endpoint is no longer managing the RX buffer for Completions through the flow control mechanism, the Application Layer must manage the RX buffer by the rate at which it issues read requests.

To determine the appropriate settings for the amount of space to reserve for completions in the RX buffer, you must make an assumption about the length of time until read completions are returned. This assumption can be estimated in terms of an additional delay, beyond the FC Update Loop Delay, as discussed in the section *Throughput of Posted Writes*. The paths for the read requests and the completions are not exactly the same as those for the posted writes and FC Updates in the PCI Express logic. However, the delay differences are probably small compared with the inaccuracy in the estimate of the external read to completion delays.

With multiple completions, the number of available credits for completion headers must be larger than the completion data space divided by the maximum packet size. Instead, the credit space for headers must be the completion data space (in bytes) divided by 64, because this is the smallest possible read completion boundary. Setting the **RX Buffer space allocation—Desired performance for received completions** to **High** under the **System Settings** heading when specifying parameter settings configures the RX buffer with enough space to meet this requirement. You can adjust this setting up or down from the **High** setting to tailor the RX buffer size to your delays and required performance.

You can also control the maximum amount of outstanding read request data. This amount is limited by the number of header tag values that can be issued by the Application Layer and by the maximum read request size that can be issued. The number of header tag values that can be in use is also limited by the IP core. You can specify 32 or 64 tags though configuration software to restrict the Application Layer to use only 32 tags. In commercial PC systems, 32 tags are usually sufficient to maintain optimal read throughput.

Design Implementation 12

2014.12.15

UG-01110_avst

Subscribe Send Feedback

Completing your design includes additional steps to specify analog properties, pin assignments, and timing constraints.

Making Analog QSF Assignments Using the Assignment Editor

You specify the analog parameters using the Quartus II Assignment Editor, the Pin Planner, or through the Quartus II Settings File **.(qsf)**.

Table 12-1: Power Supply Voltage Requirements

Data Rate	V_{CCR_GXB} and V_{CCT_GXB}	V_{CCA_GXB}
Cyclone V GX: Gen1 and Gen2	1.1 V	2.5 V
Cyclone V GT: Gen1 and Gen2	1.2 V	2.5 V

The Quartus II software provides default values for analog parameters. You can change the defaults using the Assignment Editor or the Pin Planner. You can also edit your **.qsf** directly or by typing commands in the Quartus II Tcl Console.

The following example shows how to change the value of the voltages required:

1. On the Assignments menu, select **Assignment Editor**. The Assignment Editor appears.
2. Complete the following steps for each pin requiring the V_{CCR_GXB} and V_{CCT_GXB} voltage:
   a. Double-click in the **Assignment Name** column and scroll to the bottom of the available assignments.
   b. Select **VCCR_GXB/VCCT_GXB Voltage**.
   c. In the **Value** column, select **1_1V** from the list.
3. Complete the following steps for each pin requiring the V_{CCA_GXB} voltage:
   a. Double-click in the **Assignment Name** column and scroll to the bottom of the available assignments.
   b. Select **VCCA_GXB Voltage**.
   c. In the **Value** column, select **3_0V** from the list.

The Quartus II software adds these instance assignments commands to the **.qsf** file for your project.

© 2014 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

You can also enter these commands at the Quartus II Tcl Console. For example, the following command sets the XCVR_VCCR_VCCT_VOLTAGE to 1.0 V for the pin specified:

```
set_instance_assignment -name XCVR_VCCR_VCCT_VOLTAGE 1_0V to “pin”
```

Related Information

- **Cyclone V Device Family Pin Connection Guidelines**
- **Cyclone V Device Datasheet**

Making Pin Assignments to Assign I/O Standard to Serial Data Pins

Before running Quartus II compilation, use the **Pin Planner** to assign I/O standards to the pins of the device.

1. On the Quartus II **Assignments** menu, select **Pin Planner**.
   The **Pin Planner** appears.
2. In the **Node Name** column, locate the PCIe serial data pins.
3. In the **I/O Standard** column, double-click the right-hand corner of the box to bring up a list of available I/O standards.
4. Select the pseudo current mode logic standard, **1.5 V PCML I/O**.

The Quartus II software adds instance assignments to your Quartus II Settings File (***.qsf**). The assignment is in the form `set_instance_assignment -name IO_STANDARD <"IO_STANDARD_NAME"> -to <signal_name>`. The ***.qsf** is in your synthesis directory.

Related Information

Arria 10 GX GT, and SX Device Family Pin Connection Guidelines

Recommended Reset Sequence to Avoid Link Training Issues

1. Wait until the FPGA is configured as indicated by the assertion of CONFIG_DONE from the FPGA block controller.
2. Deassert the mgmt_rst_reset input to the Transceiver Reconfiguration Controller IP Core.
3. Wait for tx_cal_busy and rx_cal_busy SERDES outputs to be deasserted.
4. Deassert pin_perstn to take the Hard IP for PCIe out of reset. For plug-in cards, the minimum assertion time for pin_perstn is 100 ms. Embedded systems do not have a minimum assertion time for pin_perstn.
5. Wait for thereset_status output to be deasserted
6. Deassert the reset output to the Application Layer.

Related Information

Reset Sequence for Hard IP for PCI Express IP Core and Application Layer on page 6-3

Optional Features 13

2014.12.15

UG-01110_avst Subscribe Send Feedback

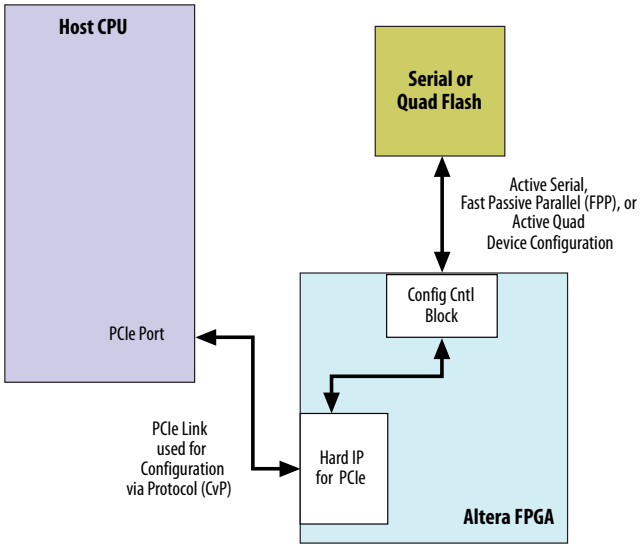
Configuration via Protocol (CvP)

The Hard IP for PCI Express architecture has an option to configure the FPGA and initialize the PCI Express link. In prior devices, a single Program Object File (**.pof**) programmed the I/O ring and FPGA fabric before the PCIe link training and enumeration began. The **.pof** file is divided into two parts:

- The I/O bitstream contains the data to program the I/O ring, the Hard IP for PCI Express, and other elements that are considered part of the periphery image.
- The core bitstream contains the data to program the FPGA fabric.

When you select the CvP design flow, the I/O ring and PCI Express link are programmed first, allowing the PCI Express link to reach the L0 state and begin operation independently, before the rest of the core is programmed. After the PCI Express link is established, it can be used to program the rest of the device. The following figure shows the blocks that implement CvP.

Figure 13-1: CvP in Cyclone V Devices


Host CPU
Serial or Quad Flash
Active Serial, Fast Passive Parallel (FPP), or Active Quad Device Configuration
Config Cntl Block
PCIe Port
PCIe Link used for Configuration via Protocol (CvP)
Hard IP for PCIe
Altera FPGA

© 2014 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

CvP has the following advantages:

- Provides a simpler software model for configuration. A smart host can use the PCIe protocol and the application topology to initialize and update the FPGA fabric.
- Enables dynamic core updates without requiring a system power down.
- Improves security for the proprietary core bitstream.
- Reduces system costs by reducing the size of the flash device to store the **.pof**.
- Facilitates hardware acceleration.
- May reduce system size because a single CvP link can be used to configure multiple FPGAs.

Table 13-1: CvP Support

CvP is available for the following configurations.

Data Rate and Application Interface Width	Support
Gen1 128-bit interface to Application Layer	Supported
Gen2 128-bit interface to Application Layer	Contact your Altera sales representative

Note: You cannot use dynamic transceiver reconfiguration for the transceiver channels in the CvP-enabled Hard IP when CvP is enabled.

Related Information

Configuration via Protocol (CvP) Implementation in Altera FPGAs User Guide

ECRC

ECRC ensures end-to-end data integrity for systems that require high reliability. You can specify this option under the **Error Reporting** heading. The ECRC function includes the ability to check and generate ECRC. In addition, the ECRC function can forward the TLP with ECRC to the RX port of the Application Layer. When using ECRC forwarding mode, the ECRC check and generation are performed in the Application Layer.

You must turn on **Advanced error reporting (AER)**, **ECRC checking**, and **ECRC generation** under the **PCI Express/PCI Capabilities** heading using the parameter editor to enable this functionality.

For more information about error handling, refer to *Error Signaling and Logging* in Section 6.2 of the *PCI Express Base Specification*.

ECRC on the RX Path

When the **ECRC generation** option is turned on, errors are detected when receiving TLPs with a bad ECRC. If the **ECRC generation** option is turned off, no error detection occurs. If the **ECRC forwarding** option is turned on, the ECRC value is forwarded to the Application Layer with the TLP. If the **ECRC forwarding** option is turned off, the ECRC value is not forwarded.

Table 13-2: ECRC Operation on RX Path

ECRC Forwarding	ECRC Check Enable (6)	ECRC Status	Error	TLP Forward to Application Layer
No	No	none	No	Forwarded
		good	No	Forwarded without its ECRC
		bad	No	Forwarded without its ECRC
	Yes	none	No	Forwarded
		good	No	Forwarded without its ECRC
		bad	Yes	Not forwarded
Yes	No	none	No	Forwarded
		good	No	Forwarded with its ECRC
		bad	No	Forwarded with its ECRC
	Yes	none	No	Forwarded
		good	No	Forwarded with its ECRC
		bad	Yes	Not forwarded

ECRC on the TX Path

When the **ECRC generation** option is on, the TX path generates ECRC. If you turn on **ECRC forwarding**, the ECRC value is forwarded with the TLP. The following table summarizes the TX ECRC generation and forwarding. All unspecified cases are unsupported and the behavior of the Hard IP is unknown.In this table, if `TD` is 1, the TLP includes an ECRC. `TD` is the TL digest bit of the TL packet described in Appendix A, Transaction Layer Packet (TLP) Header Formats.

(6) The `ECRC Check Enable` field is in the `Configuration Space Advanced Error Capabilities and Control Register`.

Table 13-3: ECRC Generation and Forwarding on TX Path

All unspecified cases are unsupported and the behavior of the Hard IP is unknown.

ECRC Forwarding	ECRC Generation Enable [7]	TLP on Application	TLP on Link	Comments
No	No	TD=0, without ECRC	TD=0, without ECRC	
		TD=1, without ECRC	TD=0, without ECRC	
	Yes	TD=0, without ECRC	TD=1, with ECRC	ECRC is generated
		TD=1, without ECRC	TD=1, with ECRC	
Yes	No	TD=0, without ECRC	TD=0, without ECRC	Core forwards the ECRC
		TD=1, with ECRC	TD=1, with ECRC	
	Yes	TD=0, without ECRC	TD=0, without ECRC	
		TD=1, with ECRC	TD=1, with ECRC	

Related Information

Transaction Layer Packet (TLP) Header Formats on page 18-1

[7] The ECRC Generation Enable field is in the Configuration Space Advanced Error Capabilities and Control Register.

Hard IP Reconfiguration 14

2014.12.15

UG-01110_avst

Subscribe Send Feedback

The Cyclone V Hard IP for PCI Express reconfiguration block allows you to dynamically change the value of configuration registers that are *read-only*. You access this block using its Avalon-MM slave interface. You must enable this optional functionality by turning on **Enable Hard IP Reconfiguration** in the parameter editor. For a complete description of the signals in this interface, refer to *Hard IP Reconfiguration Interface*.

The Hard IP reconfiguration block provides access to *read-only* configuration registers, including Configuration Space, Link Configuration, MSI and MSI-X capabilities, Power Management, and Advanced Error Reporting (AER). This interface does not support simulation.

The procedure to dynamically reprogram these registers includes the following three steps:

1. Bring down the PCI Express link by asserting the `hip_reconfig_rst_n` reset signal, if the link is already up. (Reconfiguration can occur before the link has been established.)
2. Reprogram configuration registers using the Avalon-MM slave Hard IP reconfiguration interface.
3. Release the `npor` reset signal.

Note: You can use the LMI interface to change the values of configuration registers that are *read/write* at run time. For more information about the LMI interface, refer to *LMI Signals*.

Contact your Altera representative for descriptions of the read-only, reconfigurable registers.

Related Information

LMI Signals on page 4-34

© 2014 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

Transceiver PHY IP Reconfiguration 15

2014.12.15

UG-01110_avst

Subscribe Send Feedback

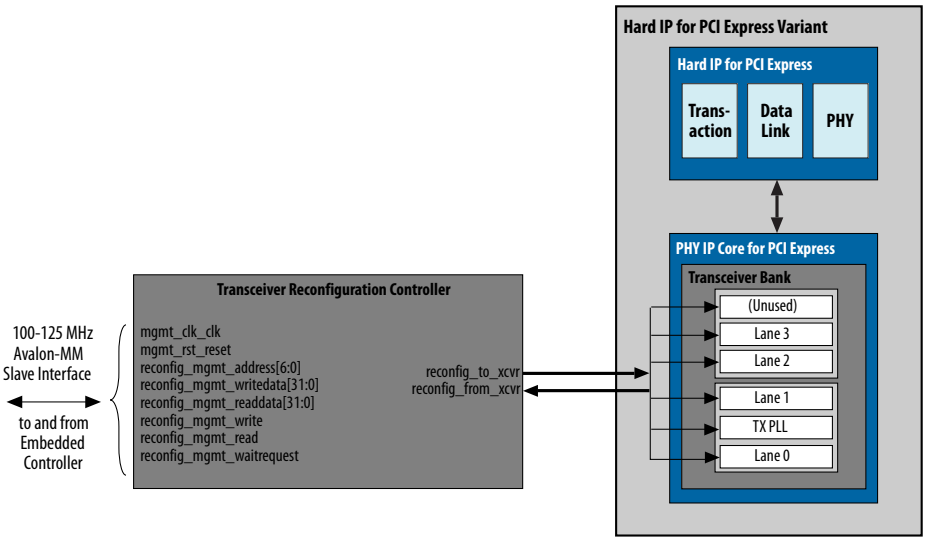
As silicon progresses towards smaller process nodes, circuit performance is affected by variations due to process, voltage, and temperature (PVT). Designs typically require offset cancellation to ensure correct operation. At Gen2 data rates, designs also require DCD calibration. Altera's Qsys example designs all include Transceiver Reconfiguration Controller and Altera PCIe Reconfig Driver IP cores to perform these functions.

Connecting the Transceiver Reconfiguration Controller IP Core

The Transceiver Reconfiguration Controller IP Core is available for V-series devices and can be found in the **Interface Protocols/Transceiver PHY** category in the IP Catalog. When you instantiate the Transceiver Reconfiguration Controller the **Enable offset cancellation block** and **Enable PLL calibration** options are enabled by default.

Figure 15-1: Altera Transceiver Reconfiguration Controller Connectivity

The following figure shows the connections between the Transceiver Reconfiguration Controller instance and the PHY IP Core for PCI Express instance for a ×4 variant.



© 2014 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

As this figure illustrates, the `reconfig_to_xcvr[ <n> 70-1:0]` and `reconfig_from_xcvr[ <n> 46-1:0]` buses connect the two components. You must provide a 100–125 MHz free-running clock to the `mgmt_clk_clk` clock input of the Transceiver Reconfiguration Controller IP Core.

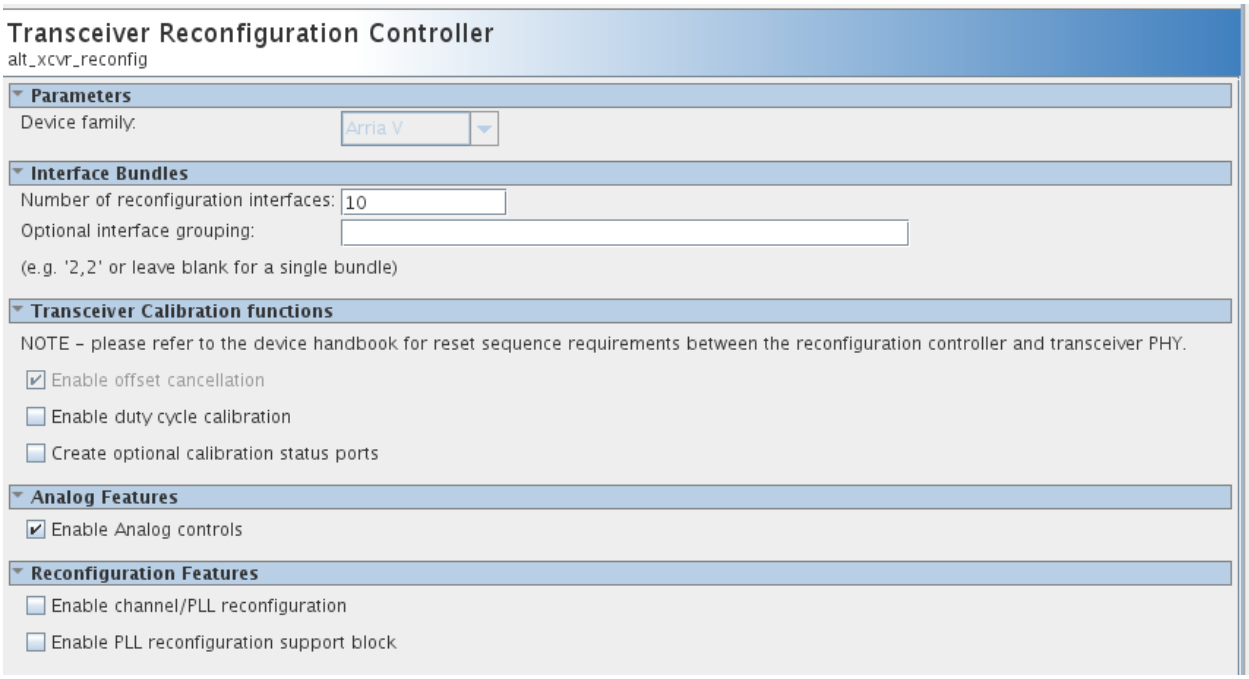
Initially, each lane and TX PLL require a separate reconfiguration interface. The parameter editor reports this number in the message pane. You must take note of this number so that you can enter it as a parameter value in the Transceiver Reconfiguration Controller parameter editor. The following figure illustrates the messages reported for a Gen2 ×4 variant. The variant requires five interfaces: one for each lane and one for the TX PLL.

Figure 15-2: Number of External Reconfiguration Controller Interfaces

ep_g2x4.DUT	5 reconfiguration interfaces are required for connection to the external reconfiguration controller and the reconfig driver
ep_g2x4.DUT	Credit allocation in the 16 KBytes receive buffer:
ep_g2x4.DUT	Posted : header=16 data=16
ep_g2x4.DUT	Non posted: header=16 data=0
ep_g2x4.DUT	Completion: header=195 data=781

When you instantiate the Transceiver Reconfiguration Controller, you must specify the required **Number of reconfiguration interfaces** as the following figure illustrates.

Figure 15-3: Specifying the Number of Transceiver Interfaces for Arria V and Cyclone V Devices



The Transceiver Reconfiguration Controller includes an **Optional interface grouping** parameter. Transceiver banks include six channels. For a ×4 variant, no special interface grouping is required because all 4 lanes and the TX PLL fit in one bank.

Note: Although you must initially create a separate logical reconfiguration interface for each lane and TX PLL in your design, when the Quartus II software compiles your design, it reduces the original

number of logical interfaces by merging them. Allowing the Quartus II software to merge reconfiguration interfaces gives the Fitter more flexibility in placing transceiver channels.

Note: You cannot use SignalTap to observe the reconfiguration interfaces.

Transceiver Reconfiguration Controller Connectivity for Designs Using CvP

If your design meets the following criteria:

- It enables CvP
- It includes an additional transceiver PHY that connect to the same Transceiver Reconfiguration Controller

then you must connect the PCIe `refclk` signal to the `mgmt_clk_clk` signal of the Transceiver Reconfiguration Controller and the additional transceiver PHY. In addition, if your design includes more than one Transceiver Reconfiguration Controller on the same side of the FPGA, they all must share the `mgmt_clk_clk` signal.

For more information about using the Transceiver Reconfiguration Controller, refer to the *Transceiver Reconfiguration Controller* chapter in the *Altera Transceiver PHY IP Core User Guide*.

Related Information

Altera Transceiver PHY IP Core User Guide

Testbench and Design Example 16

2014.12.15

UG-01110_avst

Subscribe Send Feedback

This chapter introduces the Root Port or Endpoint design example including a testbench, BFM, and a test driver module. You can create this design example for using design flows described in *Getting Started with the Cyclone V Hard IP for PCI Express* .

When configured as an Endpoint variation, the testbench instantiates a design example and a Root Port BFM, which provides the following functions:

- A configuration routine that sets up all the basic configuration registers in the Endpoint. This configuration allows the Endpoint application to be the target and initiator of PCI Express transactions.
- A Verilog HDL procedure interface to initiate PCI Express transactions to the Endpoint.

The testbench uses a test driver module, **altpcietb_bfm_driver_chaining** to exercise the chaining DMA of the design example. The test driver module displays information from the Endpoint Configuration Space registers, so that you can correlate to the parameters you specified using the parameter editor.

When configured as a Root Port, the testbench instantiates a Root Port design example and an Endpoint model, which provides the following functions:

- A configuration routine that sets up all the basic configuration registers in the Root Port and the Endpoint BFM. This configuration allows the Endpoint application to be the target and initiator of PCI Express transactions.
- A Verilog HDL procedure interface to initiate PCI Express transactions to the Endpoint BFM.

The testbench uses a test driver module, **altpcietb_bfm_driver_rp**, to exercise the target memory and DMA channel in the Endpoint BFM. The test driver module displays information from the Root Port Configuration Space registers, so that you can correlate to the parameters you specified using the parameter editor. The Endpoint model consists of an Endpoint variation combined with the chaining DMA application described above.

Note: The Altera testbench and Root Port or Endpoint BFM provide a simple method to do basic testing of the Application Layer logic that interfaces to the variation. However, the testbench and Root Port BFM are not intended to be a substitute for a full verification environment. To thoroughly test your application, Altera suggests that you obtain commercially available PCI Express verification IP and tools, or do your own extensive hardware testing or both.

© 2014 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

Your Application Layer design may need to handle at least the following scenarios that are not possible to create with the Altera testbench and the Root Port BFM:

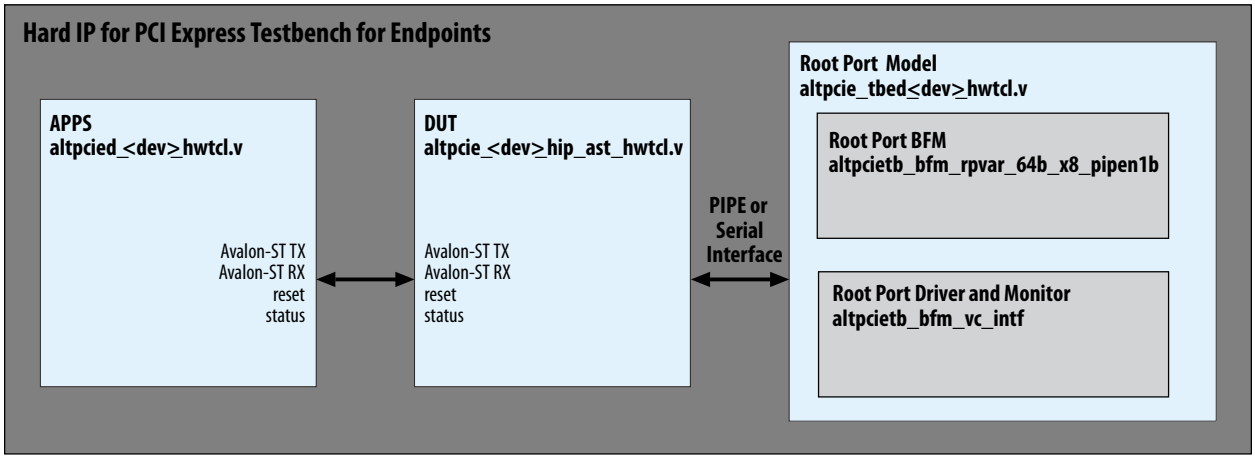
- It is unable to generate or receive Vendor Defined Messages. Some systems generate Vendor Defined Messages and the Application Layer must be designed to process them. The Hard IP block passes these messages on to the Application Layer which, in most cases should ignore them.
- It can only handle received read requests that are less than or equal to the currently set **Maximum payload size** option specified under **PCI Express/PCI Capabilities** heading under the **Device** tab using the parameter editor. Many systems are capable of handling larger read requests that are then returned in multiple completions.
- It always returns a single completion for every read request. Some systems split completions on every 64-byte address boundary.
- It always returns completions in the same order the read requests were issued. Some systems generate the completions out-of-order.
- It is unable to generate zero-length read requests that some systems generate as flush requests following some write transactions. The Application Layer must be capable of generating the completions to the zero length read requests.
- It uses fixed credit allocation.
- It does not support parity.
- It does not support multi-function designs which are available when using Configuration Space Bypass mode or Single Root I/O Virtualization (SR-IOV).

Endpoint Testbench

After you install the Quartus II software, you can copy any of the example designs from the ***<install_dir>*/ip/altera/altera_pcie/altera_pcie_hip_ast_ed/example_design** directory. You can generate the testbench from the example design as was shown in *Getting Started with the Cyclone V Hard IP for PCI Express*.

This testbench simulates up to an ×8 PCI Express link using either the PIPE interfaces of the Root Port and Endpoints or the serial PCI Express interface. The testbench design does not allow more than one PCI Express link to be simulated at a time. The following figure presents a high level view of the design example.

Figure 16-1: Design Example for Endpoint Designs



The top-level of the testbench instantiates four main modules:

- <qsys_systemname>— This is the example Endpoint design. For more information about this module, refer to *Chaining DMA Design Examples*.
- **altpcietb_bfm_top_rp.v**—This is the Root Port PCI Express BFM. For more information about this module, refer to *Root Port BFM*.
- **altpcietb_pipe_phy**—There are eight instances of this module, one per lane. These modules interconnect the PIPE MAC layer interfaces of the Root Port and the Endpoint. The module mimics the behavior of the PIPE PHY layer to both MAC interfaces.
- **altpcietb_bfm_driver_chaining**—This module drives transactions to the Root Port BFM. This is the module that you modify to vary the transactions sent to the example Endpoint design or your own design. For more information about this module, refer to *Root Port Design Example*.

In addition, the testbench has routines that perform the following tasks:

- Generates the reference clock for the Endpoint at the required frequency.
- Provides a PCI Express reset at start up.

Note: Before running the testbench, you should set the following parameters:

- `serial_sim_hwtcl`: Set this parameter in **<instantiation name>_tb.v** . This parameter controls whether the testbench simulates in PIPE mode or serial mode. When is set to 0, the simulation runs in PIPE mode; when set to 1, it runs in serial mode. Although the `serial_sim_hwtcl` parameter is available in other files, if you set this parameter at the lower level, then it will get overwritten by the **tb.v** level.
- `serial_sim_hwtcl`: Set to 1 for serial simulation and 0 for PIPE simulation.
- `enable_pipe32_sim_hwtcl`: Set to 0 for serial simulation and 1 for PIPE simulation.

Related Information

- **Getting Started with the Cyclone V Hard IP for PCI Express**
- **Chaining DMA Design Examples** on page 16-4
- **Root Port Testbench** on page 16-4
- **Root Port Design Example** on page 16-21

Root Port Testbench

This testbench simulates up to an ×8 PCI Express link using either the PIPE interfaces of the Root Port and Endpoints or the serial PCI Express interface. The testbench design does not allow more than one PCI Express link to be simulated at a time. The top-level of the testbench instantiates four main modules:

- *<qsys_systemname>*— Name of Root Port This is the example Root Port design. For more information about this module, refer to *Root Port Design Example*.
- **altpcietb_bfm_ep_example_chaining_pipen1b**—This is the Endpoint PCI Express mode described in the section *Chaining DMA Design Examples*.
- **altpcietb_pipe_phy**—There are eight instances of this module, one per lane. These modules connect the PIPE MAC layer interfaces of the Root Port and the Endpoint. The module mimics the behavior of the PIPE PHY layer to both MAC interfaces.
- **altpcietb_bfm_driver_rp**—This module drives transactions to the Root Port BFM. This is the module that you modify to vary the transactions sent to the example Endpoint design or your own design. For more information about this module, see *Test Driver Module*.

The testbench has routines that perform the following tasks:

- Generates the reference clock for the Endpoint at the required frequency.
- Provides a reset at start up.

Note: Before running the testbench, you should set the following parameters:

- `serial_sim_hwtcl`: Set this parameter in **<instantiation name>_tb.v** . This parameter controls whether the testbench simulates in PIPE mode or serial mode. When is set to 0, the simulation runs in PIPE mode; when set to 1, it runs in serial mode. Although the `serial_sim_hwtcl` parameter is available in other files, if you set this parameter at the lower level, then it will get overwritten by the **tb.v** level.
- `serial_sim_hwtcl`: Set to 1 for serial simulation and 0 for PIPE simulation.
- `enable_pipe32_sim_hwtcl`: Set to 0 for serial simulation and 1 for PIPE simulation.

Chaining DMA Design Examples

This design examples shows how to create a chaining DMA native Endpoint which supports simultaneous DMA read and write transactions. The write DMA module implements write operations from the Endpoint memory to the root complex (RC) memory. The read DMA implements read operations from the RC memory to the Endpoint memory.

When operating on a hardware platform, the DMA is typically controlled by a software application running on the root complex processor. In simulation, the generated testbench, along with this design example, provides a BFM driver module in Verilog HDL that controls the DMA operations. Because the example relies on no other hardware interface than the PCI Express link, you can use the design example for the initial hardware validation of your system.

The design example includes the following two main components:

- The Root Port variation
- An Application Layer design example

The end point or Root Port variant is generated in the language (Verilog HDL or VHDL) that you selected for the variation file. The testbench files are only generated in Verilog HDL in the current release. If you choose to use VHDL for your variant, you must have a mixed-language simulator to run this testbench.

Note: The chaining DMA design example requires setting BAR 2 or BAR 3 to a minimum of 256 bytes. To run the DMA tests using MSI, you must set the **Number of MSI messages requested** parameter under the **PCI Express/PCI Capabilities** page to at least 2.

The chaining DMA design example uses an architecture capable of transferring a large amount of fragmented memory without accessing the DMA registers for every memory block. For each block of memory to be transferred, the chaining DMA design example uses a descriptor table containing the following information:

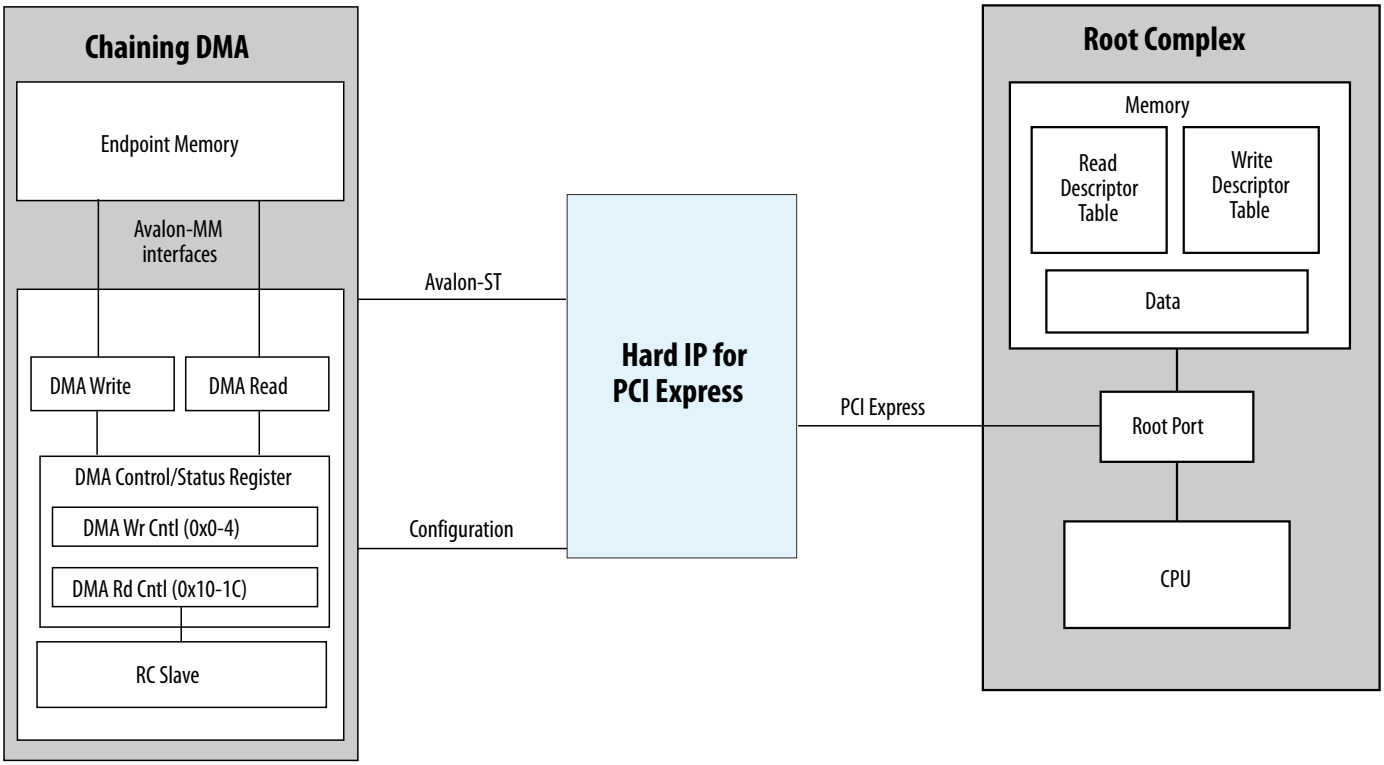
- Length of the transfer
- Address of the source
- Address of the destination
- Control bits to set the handshaking behavior between the software application or BFM driver and the chaining DMA module

Note: The chaining DMA design example only supports dword-aligned accesses. The chaining DMA design example does not support ECRC forwarding.

The BFM driver writes the descriptor tables into BFM shared memory, from which the chaining DMA design engine continuously collects the descriptor tables for DMA read, DMA write, or both. At the beginning of the transfer, the BFM programs the Endpoint chaining DMA control register. The chaining DMA control register indicates the total number of descriptor tables and the BFM shared memory address of the first descriptor table. After programming the chaining DMA control register, the chaining DMA engine continuously fetches descriptors from the BFM shared memory for both DMA reads and DMA writes, and then performs the data transfer for each descriptor.

The following figure shows a block diagram of the design example connected to an external RC CPU. For a description of the DMA write and read registers, *Chaining DMA Control and Status Registers*.

Figure 16-2: Top-Level Chaining DMA Example for Simulation



The block diagram contains the following elements:

- Endpoint DMA write and read requester modules.
- The chaining DMA design example connects to the Avalon-ST interface of the Cyclone V Hard IP for PCI Express. The connections consist of the following interfaces:
  - The Avalon-ST RX receives TLP header and data information from the Hard IP block
  - The Avalon-ST TX transmits TLP header and data information to the Hard IP block
  - The Avalon-ST MSI port requests MSI interrupts from the Hard IP block
  - The sideband signal bus carries static information such as configuration information
- The descriptor tables of the DMA read and the DMA write are located in the BFM shared memory.
- A RC CPU and associated PCI Express PHY link to the Endpoint design example, using a Root Port and a north/south bridge.

The example Endpoint design Application Layer accomplishes the following objectives:

- Shows you how to interface to the Cyclone V Hard IP for PCI Express using the Avalon-ST protocol.
- Provides a chaining DMA channel that initiates memory read and write transactions on the PCI Express link.
- If the ECRC forwarding functionality is enabled, provides a CRC Compiler IP core to check the ECRC dword from the Avalon-ST RX path and to generate the ECRC for the Avalon-ST TX path.

The following modules are included in the design example and located in the subdirectory **<*qsys_systemname*>/testbench/<*qsys_system_name*>_tb/simulation/submodules** :

- <*qsys_systemname*> —This module is the top level of the example Endpoint design that you use for simulation.

  This module provides both PIPE and serial interfaces for the simulation environment. This module has a `test_in` debug ports. Refer to *Test Signals*which allow you to monitor and control internal states of the Hard IP.

  For synthesis, the top level module is **<*qsys_systemname*>/synthesis/submodules**. This module instantiates the top-level module and propagates only a small sub-set of the test ports to the external I/Os. These test ports can be used in your design.
- <*variation name*>**.v** or <*variation name*>**.vhd**— Because Altera provides five sample parameterizations, you may have to edit one of the provided examples to create a simulation that matches your requirements. <*variation name*>**.v** or <*variation name*>**.vhd**— Because Altera provides five sample parameterizations, you may have to edit one of the provided examples to create a simulation that matches your requirements.

The chaining DMA design example hierarchy consists of these components:

- A DMA read and a DMA write module
- An on-chip Endpoint memory (Avalon-MM slave) which uses two Avalon-MM interfaces for each engine

The RC slave module is used primarily for downstream transactions which target the Endpoint on-chip buffer memory. These target memory transactions bypass the DMA engines. In addition, the RC slave module monitors performance and acknowledges incoming message TLPs. Each DMA module consists of these components:

- Control register module—The RC programs the control register (four dwords) to start the DMA.
- Descriptor module—The DMA engine fetches four dword descriptors from BFM shared memory which hosts the chaining DMA descriptor table.
- Requester module—For a given descriptor, the DMA engine performs the memory transfer between Endpoint memory and the BFM shared memory.

The following modules are provided in both Verilog HDL:

- **altpcierd_example_app_chaining**—This top level module contains the logic related to the Avalon-ST interfaces as well as the logic related to the sideband bus. This module is fully register bounded and can be used as an incremental re-compile partition in the Quartus II compilation flow.
- **altpcierd_cdma_ast_rx**, **altpcierd_cdma_ast_rx_64**, **altpcierd_cdma_ast_rx_128**—These modules implement the Avalon-ST receive port for the chaining DMA. The Avalon-ST receive port converts the Avalon-ST interface of the IP core to the descriptor/data interface used by the chaining DMA submodules. **altpcierd_cdma_ast_rx** is used with the descriptor/data IP core (through the ICM). **altpcierd_cdma_ast_rx_64** is used with the 64-bit Avalon-ST IP core. **altpcierd_cdma_ast_rx_128** is used with the 128-bit Avalon-ST IP core.
- **altpcierd_cdma_ast_tx**, **altpcierd_cdma_ast_tx_64**, **altpcierd_cdma_ast_tx_128**—These modules implement the Avalon-ST transmit port for the chaining DMA. The Avalon-ST transmit port converts the descriptor/data interface of the chaining DMA submodules to the Avalon-ST interface of the IP core. **altpcierd_cdma_ast_tx** is used with the descriptor/data IP core (through the ICM). **altpcierd_cdma_ast_tx_64** is used with the 64-bit Avalon-ST IP core. **altpcierd_cdma_ast_tx_128** is used with the 128-bit Avalon-ST IP core.
- **altpcierd_cdma_ast_msi**—This module converts MSI requests from the chaining DMA submodules into Avalon-ST streaming data.
- **alpcierd_cdma_app_icm**—This module arbitrates PCI Express packets for the modules **altpcierd_dma_dt** (read or write) and **altpcierd_rc_slave**. **alpcierd_cdma_app_icm** instantiates the Endpoint memory used for the DMA read and write transfer.
- **alt pcierd_compliance_test.v**—This module provides the logic to perform CBB via a push button.
- **altpcierd_rc_slave**—This module provides the completer function for all downstream accesses. It instantiates the **altpcierd_rxtx_downstream_intf** and **altpcierd_reg_ access** modules. Downstream requests include programming of chaining DMA control registers, reading of DMA status registers, and direct read and write access to the Endpoint target memory, bypassing the DMA.
- **altpcierd_rx_tx_downstream_intf**—This module processes all downstream read and write requests and handles transmission of completions. Requests addressed to BARs 0, 1, 4, and 5 access the chaining DMA target memory space. Requests addressed to BARs 2 and 3 access the chaining DMA control and status register space using the **altpcierd_reg_access** module.
- **altpcierd_reg_access**—This module provides access to all of the chaining DMA control and status registers (BAR 2 and 3 address space). It provides address decoding for all requests and multiplexing for completion data. All registers are 32-bits wide. Control and status registers include the control registers in the **altpcierd_dma_prg_reg** module, status registers in the **altpcierd_read_dma_requester** and **altpcierd_write_dma_requester** modules, as well as other miscellaneous status registers.
- **altpcierd_dma_dt**—This module arbitrates PCI Express packets issued by the submodules **altpcierd_dma_prg_reg**, **altpcierd_read_dma_requester**, **altpcierd_write_dma_requester** and **altpcierd_dma_descriptor**.
- **al tpcierd_dma_prg_reg**—This module contains the chaining DMA control registers which get programmed by the software application or BFM driver.
- **altpcierd_dma_descriptor**—This module retrieves the DMA read or write descriptor from the BFM shared memory, and stores it in a descriptor FIFO. This module issues upstream PCI Express TLPs of type Mrd.

- **altpcierd_read_dma_requester**, **altpcierd_read_dma_requester_128**—For each descriptor located in the **altpcierd_descriptor FIFO**, this module transfers data from the BFM shared memory to the Endpoint memory by issuing MRd PCI Express transaction layer packets.
  altpcierd_read_dma_requester is used with the 64-bit Avalon-ST IP core.
  altpcierd_read_dma_requester_128 is used with the 128-bit Avalon-ST IP core.
- **altpcie rd_write_dma_requester, altpcierd_write_dma_requester_128**—For each descriptor located in the **altpcierd_descriptor FIFO**, this module transfers data from the Endpoint memory to the BFM shared memory by issuing MWr PCI Express transaction layer packets. **altpc ierd_write_dma_requester** is used with the 64-bit Avalon-ST IP core.
  altpcierd_write_dma_requester_128 is used with the 128-bit Avalon-ST IP core.ls
- **altpcierd_cpld_rx_buffer**—This modules monitors the available space of the RX Buffer; It prevents RX Buffer overflow by arbitrating memory read request issued by the application.
- **altpcierd_cplerr_lmi**—This module transfers the err_desc_func0 from the application to the Hard IP block using the LMI interface. It also retimes the `cpl_err` bits from the application to the Hard IP block.
- **altpcierd_tl_cfg_sample**—This module demultiplexes the Configuration Space signals from the `tl_cfg_ctl` bus from the Hard IP block and synchronizes this information, along with the `tl_cfg_sts` bus to the user clock (`pld_clk`) domain.

Related Information

- **Test Signals**
- **Chaining DMA Control and Status Registers** on page 16-10

BAR/Address Map

The design example maps received memory transactions to either the target memory block or the control register block based on which BAR the transaction matches. There are multiple BARs that map to each of these blocks to maximize interoperability with different variation files. The following table shows the mapping.

Table 16-1: BAR Map

Memory BAR	Mapping
32-bit BAR0 32-bit BAR1 64-bit BAR1:0	Maps to 32 KByte target memory block. Use the rc_slave module to bypass the chaining DMA.
32-bit BAR2 32-bit BAR3 64-bit BAR3:2	Maps to DMA Read and DMA write control and status registers, a minimum of 256 bytes.
32-bit BAR4 32-bit BAR5 64-bit BAR5:4	Maps to 32 KByte target memory block. Use the rc_slave module to bypass the chaining DMA.

Memory BAR	Mapping
Expansion ROM BAR	Not implemented by design example; behavior is unpredictable.
I/O Space BAR (any)	Not implemented by design example; behavior is unpredictable.

Chaining DMA Control and Status Registers

The software application programs the chaining DMA control register located in the Endpoint application. The following table describes the control registers which consists of four dwords for the DMA write and four dwords for the DMA read. The DMA control registers are read/write.

In this table, Addr specifies the Endpoint byte address offset from BAR2 or BAR3.

Table 16-2: Chaining DMA Control Register Definitions

Addr	Register Name	Bits[31:]24	Bit[23:16]	Bit[15:0]
0x0	`DMA Wr Cntl DW0`	Control Field		Number of descriptors in descriptor table
0x4	`DMA Wr Cntl DW1`	Base Address of the Write Descriptor Table (BDT) in the RC Memory–Upper DWORD		
0x8	`DMA Wr Cntl DW2`	Base Address of the Write Descriptor Table (BDT) in the RC Memory–Lower DWORD		
0xC	`DMA Wr Cntl DW3`	Reserved	Reserved	RCLAST–Idx of last descriptor to process
0x10	`DMA Rd Cntl DW0`	Control Field (described in the next table)		Number of descriptors in descriptor table
0x14	`DMA Rd Cntl DW1`	Base Address of the Read Descriptor Table (BDT) in the RC Memory–Upper DWORD		
0x18	`DMA Rd Cntl DW2`	Base Address of the Read Descriptor Table (BDT) in the RC Memory–Lower DWORD		
0x1C	`DMA Rd Cntl DW3`	Reserved	Reserved	RCLAST–Idx of the last descriptor to process

The following table describes the control fields of the of the DMA read and DMA write control registers.

Table 16-3: Bit Definitions for the Control Field in the DMA Write Control Register and DMA Read Control Register

Bit	Field	Description
16	Reserved	—
17	MSI_ENA	Enables interrupts of all descriptors. When 1, the Endpoint DMA module issues an interrupt using MSI to the RC when each descriptor is completed. Your software application or BFM driver can use this interrupt to monitor the DMA transfer status.
18	EPLAST_ENA	Enables the Endpoint DMA module to write the number of each descriptor back to the EPLAST field in the descriptor table.
[24:20]	MSI Number	When your RC reads the MSI capabilities of the Endpoint, these register bits map to the back-end MSI signals app_msi_num [4:0]. If there is more than one MSI, the default mapping if all the MSIs are available, is: • MSI 0 = Read • MSI 1 = Write
[30:28]	MSI Traffic Class	When the RC application software reads the MSI capabilities of the Endpoint, this value is assigned by default to MSI traffic class 0. These register bits map to the back-end signal app_msi_tc[2:0].
31	DT RC Last Sync	When 0, the DMA engine stops transfers when the last descriptor has been executed. When 1, the DMA engine loops infinitely restarting with the first descriptor when the last descriptor is completed. To stop the infinite loop, set this bit to 0.

The following table defines the DMA status registers. These registers are read only. In this table, Addr specifies the Endpoint byte address offset from BAR2 or BAR3.

Table 16-4: Chaining DMA Status Register Definitions

Addr	Register Name	Bits[31:24]	Bits[23:16]	Bits[15:0]
0x20	DMA Wr Status Hi	For field definitions refer to *Fields in the DMA Write Status High Register* below.		

Addr	Register Name	Bits[31:24]	Bits[23:16]	Bits[15:0]
0x24	DMA Wr Status Lo	Target Mem Address Width	Write DMA Performance Counter. (Clock cycles from time DMA header programmed until last descriptor completes, including time to fetch descriptors.)	
0x28	DMA Rd Status Hi	For field definitions refer to *Fields in the DMA Read Status High Register* below.		
0x2C	DMA Rd Status Lo	Max No. of Tags	Read DMA Performance Counter. The number of clocks from the time the DMA header is programmed until the last descriptor completes, including the time to fetch descriptors.	
0x30	Error Status	Reserved		Error Counter. Number of bad ECRCs detected by the Application Layer. Valid only when ECRC forwarding is enabled.

The following table describes the fields of the DMA write status register. All of these fields are read only.

Table 16-5: Fields in the DMA Write Status High Register

Bit	Field	Description
[31:28]	CDMA version	Identifies the version of the chaining DMA example design.
[27:24]	Reserved	—
[23:21]	Max payload size	The following encodings are defined: • 001 128 bytes • 001 256 bytes • 010 512 bytes • 011 1024 bytes • 100 2048 bytes
[20:17]	Reserved	—
16	Write DMA descriptor FIFO empty	Indicates that there are no more descriptors pending in the write DMA.

Bit	Field	Description
[15:0]	Write DMA EPLAS	Indicates the number of the last descriptor completed by the write DMA. For simultaneous DMA read and write transfers, EPLAST is only supported for the final descriptor in the descriptor table.

The following table describes the fields in the DMA read status high register. All of these fields are read only.

Table 16-6: Fields in the DMA Read Status High Register

Bit	Field	Description
[31:24]	Reserved	—
[23:21]	Max Read Request Size	The following encodings are defined: • 001 128 bytes • 001 256 bytes • 010 512 bytes • 011 1024 bytes • 100 2048 bytes
[20:17]	Negotiated Link Width	The following encodings are defined: • 4'b0001 ×1 • 4'b0010 ×2 • 4'b0100 ×4 • 4'b1000 ×8
16	Read DMA Descriptor FIFO Empty	Indicates that there are no more descriptors pending in the read DMA.
[15:0]	Read DMA EPLAST	Indicates the number of the last descriptor completed by the read DMA. For simultaneous DMA read and write transfers, EPLAST is only supported for the final descriptor in the descriptor table.

Chaining DMA Descriptor Tables

The following table describes the Chaining DMA descriptor table. This table is stored in the BFM shared memory. It consists of a four-dword descriptor header and a contiguous list of <*n*> four-dword descriptors. The Endpoint chaining DMA application accesses the Chaining DMA descriptor table for two reasons:

- To iteratively retrieve four-dword descriptors to start a DMA
- To send update status to the RP, for example to record the number of descriptors completed to the descriptor header

Each subsequent descriptor consists of a minimum of four dwords of data and corresponds to one DMA transfer. (A dword equals 32 bits.)

Note: The chaining DMA descriptor table should not cross a 4 KByte boundary.

Table 16-7: Chaining DMA Descriptor Table

Byte Address Offset to Base Source	Descriptor Type	Description
0x0	Descriptor Header	Reserved
0x4		Reserved
0x8		Reserved
0xC		EPLAST - when enabled by the EPLAST_ENA bit in the control register or descriptor, this location records the number of the last descriptor completed by the chaining DMA module.
0x10	Descriptor 0	Control fields, DMA length
0x14		Endpoint address
0x18		RC address upper dword
0x1C		RC address lower dword
0x20	Descriptor 1	Control fields, DMA length
0x24		Endpoint address
0x28		RC address upper dword
0x2C		RC address lower dword
. . .		
0x ..0	Descriptor <*n*>	Control fields, DMA length
0x ..4		Endpoint address
0x ..8		RC address upper dword
0x ..C		RC address lower dword

The following table shows the layout of the descriptor fields following the descriptor header.

Table 16-8: Chaining DMA Descriptor Format Map

Bits[31:22]	Bits[21:16]	Bits[15:0]
Reserved	Control Fields (refer to Table 18–9)	DMA Length
Endpoint Address		
RC Address Upper DWORD		
RC Address Lower DWORD		

The following table shows the layout of the control fields of the chaining DMA descriptor.

Table 16-9: Chaining DMA Descriptor Format Map (Control Fields)

Bits[21:18]	Bit[17]	Bit[16]
Reserved	EPLAST_ENA	MSI

Each descriptor provides the hardware information on one DMA transfer. The following table describes each descriptor field.

Table 16-10: Chaining DMA Descriptor Fields

Descriptor Field	Endpoint Access	RC Access	Description
Endpoint Address	R	R/W	A 32-bit field that specifies the base address of the memory transfer on the Endpoint site.
RC Address Upper DWORD	R	R/W	Specifies the upper base address of the memory transfer on the RC site.
RC Address Lower DWORD	R	R/W	Specifies the lower base address of the memory transfer on the RC site.
DMA Length	R	R/W	Specifies the number of DMA DWORDs to transfer.

Descriptor Field	Endpoint Access	RC Access	Description
EPLAST_ENA	R	R/W	This bit is OR'd with the EPLAST_ENA bit of the control register. When EPLAST_ENA is set, the Endpoint DMA module updates the EPLAST field of the descriptor table with the number of the last completed descriptor, in the form <*0 – n*>. Refer to **Chaining DMA Descriptor Tables** on page 16-13 for more information.
MSI_ENA	R	R/W	This bit is OR'd with the MSI bit of the descriptor header. When this bit is set the Endpoint DMA module sends an interrupt when the descriptor is completed.

Test Driver Module

The BFM driver module, **altpcietb_bfm_driver_chaining.v** is configured to test the chaining DMA example Endpoint design. The BFM driver module configures the Endpoint Configuration Space registers and then tests the example Endpoint chaining DMA channel. This file is stored in the ***<working_dir>/testbench/<variation_name>/simulation/submodules*** directory.

The BFM test driver module performs the following steps in sequence:

1. Configures the Root Port and Endpoint Configuration Spaces, which the BFM test driver module does by calling the procedure ebfm_cfg_rp_ep, which is part of **altpcietb_bfm_configure**.
2. Finds a suitable BAR to access the example Endpoint design Control Register space. Either BARs 2 or 3 must be at least a 256-byte memory BAR to perform the DMA channel test. The find_mem_bar procedure in the **altpcietb_bfm_driver_chaining** does this.
3. If a suitable BAR is found in the previous step, the driver performs the following tasks:

   a. DMA read—The driver programs the chaining DMA to read data from the BFM shared memory into the Endpoint memory. The descriptor control fields are specified so that the chaining DMA completes the following steps to indicate transfer completion:

      - The chaining DMA writes the EPLast bit of the *Chaining DMA Descriptor Table* after finishing the data transfer for the first and last descriptors.
      - The chaining DMA issues an MSI when the last descriptor has completed.

   a. DMA write—The driver programs the chaining DMA to write the data from its Endpoint memory back to the BFM shared memory. The descriptor control fields are specified so that the chaining DMA completes the following steps to indicate transfer completion:

- The chaining DMA writes the `EPLast` bit of the *Chaining DMA Descriptor Table*after completing the data transfer for the first and last descriptors.
- The chaining DMA issues an MSI when the last descriptor has completed.
- The data written back to BFM is checked against the data that was read from the BFM.
- The driver programs the chaining DMA to perform a test that demonstrates downstream access of the chaining DMA Endpoint memory.

Note: Edit this file if you want to add your own custom PCIe transactions. Insert your own custom function after the `find_mem_bar` function. You can use the functions in the *BFM Procedures and Functions* section.

Related Information

- **Chaining DMA Descriptor Tables** on page 16-13
- **BFM Procedures and Functions** on page 16-31

DMA Write Cycles

The procedure `dma_wr_test` used for DMA writes uses the following steps:

1. Configures the BFM shared memory. Configuration is accomplished with three descriptor tables described below.

Table 16-11: Write Descriptor 0

	Offset in BFM in Shared Memory	Value	Description
DW0	0x810	82	Transfer length in dwords and control bits as described in *Bit Definitions for the Control Field in the DMA Write Control Register and DMA Read Control Register*.
DW1	0x814	3	Endpoint address
DW2	0x818	0	BFM shared memory data buffer 0 upper address value
DW3	0x81c	0x1800	BFM shared memory data buffer 1 lower address value
Data Buffer 0	0x1800	Increment by 1 from 0x1515_0001	Data content in the BFM shared memory from address: 0x01800–0x1840

Table 16-12: Write Descriptor 1

	Offset in BFM Shared Memory	Value	Description
DW0	0x820	1,024	Transfer length in dwords and control bits as described in *Bit Definitions for the Control Field in the DMA Write Control Register and DMA Read Control Register* .
DW1	0x824	0	Endpoint address
DW2	0x828	0	BFM shared memory data buffer 1 upper address value
DW3	0x82c	0x2800	BFM shared memory data buffer 1 lower address value
Data Buffer 1	0x02800	Increment by 1 from 0x2525_0001	Data content in the BFM shared memory from address: 0x02800

Table 16-13: Write Descriptor 2

	Offset in BFM Shared Memory	Value	Description
DW0	0x830	644	Transfer length in dwords and control bits as described in *Bit Definitions for the Control Field in the DMA Write Control Register and DMA Read Control Register*.
DW1	0x834	0	Endpoint address
DW2	0x838	0	BFM shared memory data buffer 2 upper address value
DW3	0x83c	0x057A0	BFM shared memory data buffer 2 lower address value
Data Buffer 2	0x057A0	Increment by 1 from 0x3535_0001	Data content in the BFM shared memory from address: 0x057A0

2. Sets up the chaining DMA descriptor header and starts the transfer data from the Endpoint memory to the BFM shared memory. The transfer calls the procedure `dma_set_header` which writes four dwords, DW0:DW3, into the DMA write register module.

Table 16-14: DMA Control Register Setup for DMA Write

	Offset in DMA Control Register (BAR2)	Value	Description
DW0	0x0	3	Number of descriptors and control bits as described in *Chaining DMA Control Register Definitions*.
DW1	0x4	0	BFM shared memory descriptor table upper address value
DW2	0x8	0x800	BFM shared memory descriptor table lower address value
DW3	0xc	2	Last valid descriptor

After writing the last dword, DW3, of the descriptor header, the DMA write starts the three subsequent data transfers.

3. Waits for the DMA write completion by polling the BFM share memory location 0x80c, where the DMA write engine is updating the value of the number of completed descriptor. Calls the procedures `rcmem_poll` and `msi_poll` to determine when the DMA write transfers have completed.

Related Information

Chaining DMA Control and Status Registers on page 16-10

DMA Read Cycles

The procedure `dma_rd_test` used for DMA read uses the following three steps:

1. Configures the BFM shared memory with a call to the procedure `dma_set_rd_desc_data` which sets the following three descriptor tables. .

Table 16-15: Read Descriptor 0

	Offset in BFM Shared Memory	Value	Description
DW0	0x910	82	Transfer length in dwords and control bits as described in on page 18–15
DW1	0x914	3	Endpoint address value
DW2	0x918	0	BFM shared memory data buffer 0 upper address value
DW3	0x91c	0x8DF0	BFM shared memory data buffer 0 lower address value
Data Buffer 0	0x8DF0	Increment by 1 from 0xAAA0_0001	Data content in the BFM shared memory from address: 0x89F0

Table 16-16: Read Descriptor 1

	Offset in BFM Shared Memory	Value	Description
DW0	0x920	1,024	Transfer length in dwords and control bits as described in on page 18–15
DW1	0x924	0	Endpoint address value
DW2	0x928	10	BFM shared memory data buffer 1 upper address value
DW3	0x92c	0x10900	BFM shared memory data buffer 1 lower address value
Data Buffer 1	0x10900	Increment by 1 from 0xBBBB_0001	Data content in the BFM shared memory from address: 0x10900

Table 16-17: Read Descriptor 2

	Offset in BFM Shared Memory	Value	Description
DW0	0x930	644	Transfer length in dwords and control bits as described in on page 18–15
DW1	0x934	0	Endpoint address value
DW2	0x938	0	BFM shared memory upper address value
DW3	0x93c	0x20EF0	BFM shared memory lower address value
Data Buffer 2	0x20EF0	Increment by 1 from 0xCCCC_0001	Data content in the BFM shared memory from address: 0x20EF0

2. Sets up the chaining DMA descriptor header and starts the transfer data from the BFM shared memory to the Endpoint memory by calling the procedure `dma_set_header` which writes four dwords, DW0:DW3 into the DMA read register module.

Table 16-18: DMA Control Register Setup for DMA Read

	Offset in DMA Control Registers (BAR2)	Value	Description
DW0	0x0	3	Number of descriptors and control bits as described in *Chaining DMA Control Register Definitions*.

	Offset in DMA Control Registers (BAR2)	Value	Description
DW1	0x14	0	BFM shared memory upper address value
DW2	0x18	0x900	BFM shared memory lower address value
DW3	0x1c	2	Last descriptor written

After writing the last dword of the Descriptor header (DW3), the DMA read starts the three subsequent data transfers.

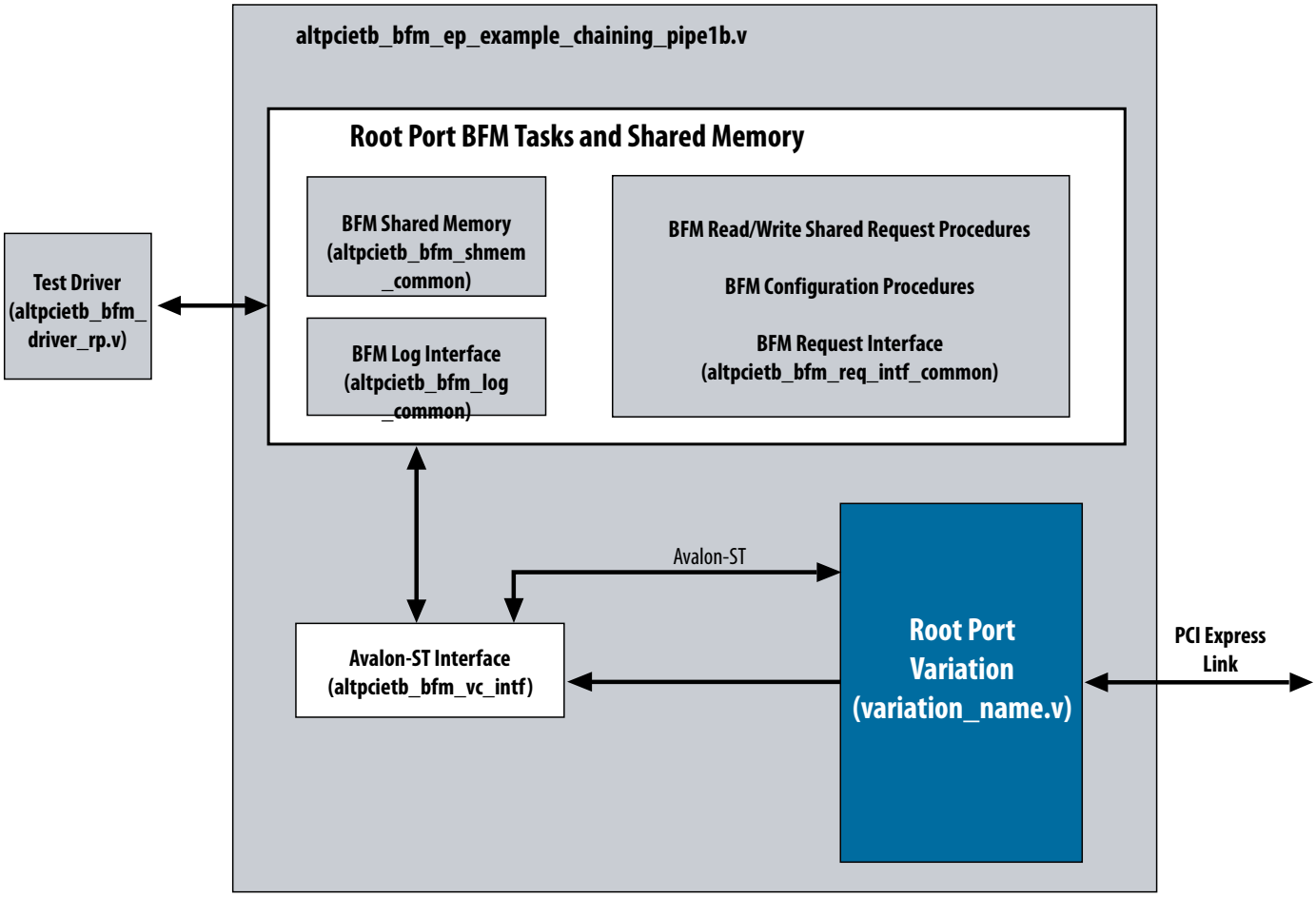
3. Waits for the DMA read completion by polling the BFM shared memory location 0x90c, where the DMA read engine is updating the value of the number of completed descriptors. Calls the procedures `rcmem_poll` and `msi_poll` to determine when the DMA read transfers have completed.

Root Port Design Example

The design example includes the following primary components:

- Root Port variation (<*qsys_systemname*>.
- Avalon-ST Interfaces (**altpcietb_bfm_vc_intf_ast**)—handles the transfer of TLP requests and completions to and from the Cyclone V Hard IP for PCI Express variation using the Avalon-ST interface.
- Root Port BFM tasks—contains the high-level tasks called by the test driver, low-level tasks that request PCI Express transfers from **altpcietb_bfm_vc_intf_ast**, the Root Port memory space, and simulation functions such as displaying messages and stopping simulation.
- Test Driver (**altpcietb_bfm_driver_rp.v**)—the chaining DMA Endpoint test driver which configures the Root Port and Endpoint for DMA transfer and checks for the successful transfer of data. Refer to the *Test Driver Module*for a detailed description.

Figure 16-3: Root Port Design Example



You can use the example Root Port design for Verilog HDL simulation. All of the modules necessary to implement the example design with the variation file are contained in **altpcietb_bfm_ep_example_chaining_pipen1b.v**.

The top-level of the testbench instantiates the following key files:

- **altlpcietb_bfm_top_ep.v**— this is the Endpoint BFM. This file also instantiates the SERDES and PIPE interface.
- **altpcietb_pipe_phy.v**—used to simulate the PIPE interface.
- **altp cietb_bfm_ep_example_chaining_pipen1b.v**—the top-level of the Root Port design example that you use for simulation. This module instantiates the Root Port variation, <*variation_name*> **.v**, and the Root Port application **altpcietb_bfm_vc_intf** _<*application_width*> . This module provides both PIPE and serial interfaces for the simulation environment. This module has two debug ports named `test_out_icm_`(which is the `test_out` signal from the Hard IP) and `test_in` which allows you to monitor and control internal states of the Hard IP variation.
- **altpcietb_bfm_vc_intf_ast.v**—a wrapper module which instantiates either **altpcietb_vc_intf_64** or **altpcietb_vc_intf_** <*application_width*> based on the type of Avalon-ST interface that is generated.
- **altpcietb_vc_intf_** _<*application_width*> **.v**—provide the interface between the Cyclone V Hard IP for PCI Express variant and the Root Port BFM tasks. They provide the same function as the **altpcietb_bfm_vc_intf.v** module, transmitting requests and handling completions. Refer to the *Root Port BFM* for a full description of this function. This version uses Avalon-ST signalling with either a 64- or 128-bit data bus interface.
- **altpcierd_tl_cfg_sample.v**—accesses Configuration Space signals from the variant. Refer to the *Chaining DMA Design Examples* for a description of this module.

Files in subdirectory <*qsys_systemname*> **/testbench/simulation/submodules**:

- **altpcietb_bfm_ep_example_chaining_pipen1b.v**—the simulation model for the chaining DMA Endpoint.
- **altpcietb_bfm_driver_rp.v**–this file contains the functions to implement the shared memory space, PCI Express reads and writes, initialize the Configuration Space registers, log and display simulation messages, and define global constants.

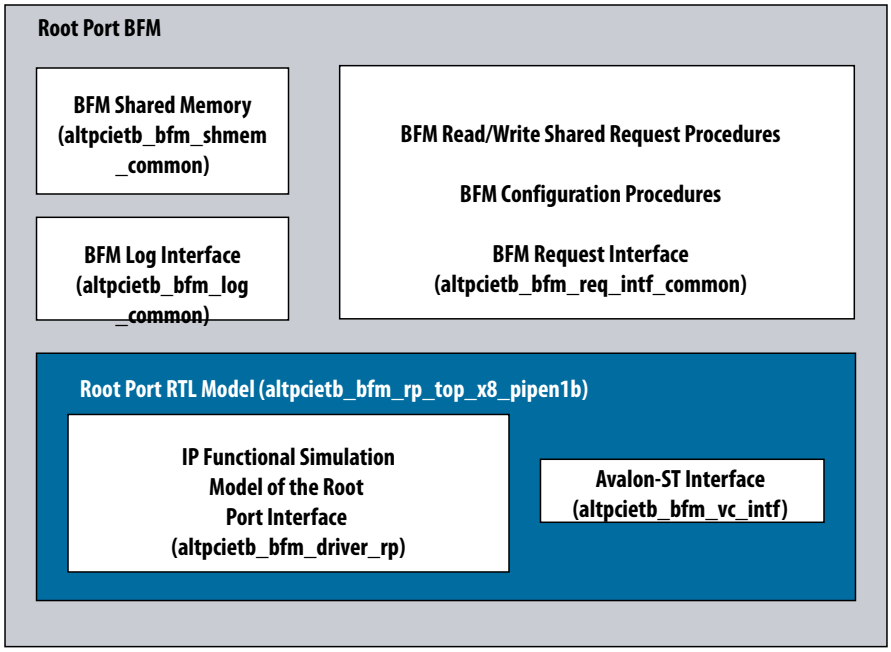
Related Information

- **Test Driver Module** on page 16-16
- **Chaining DMA Design Examples** on page 16-4

Root Port BFM

The basic Root Port BFM provides Verilog HDL task-based interface for requesting transactions that are issued to the PCI Express link. The Root Port BFM also handles requests received from the PCI Express link. The following figure provides an overview of the Root Port BFM.

Figure 16-4: Root Port BFM


Root Port BFM
BFM Shared Memory (altpcietb_bfm_shmem_common)
BFM Log Interface (altpcietb_bfm_log_common)
BFM Read/Write Shared Request Procedures
BFM Configuration Procedures
BFM Request Interface (altpcietb_bfm_req_intf_common)
Root Port RTL Model (altpcietb_bfm_rp_top_x8_pipen1b)
IP Functional Simulation Model of the Root Port Interface (altpcietb_bfm_driver_rp)
Avalon-ST Interface (altpcietb_bfm_vc_intf)

The functionality of each of the modules included is explained below.

- BFM shared memory (**altpcietb_bfm_shmem_common** Verilog HDL include file)—The Root Port BFM is based on the BFM memory that is used for the following purposes:
- Storing data received with all completions from the PCI Express link.
- Storing data received with all write transactions received from the PCI Express link.
- Sourcing data for all completions in response to read transactions received from the PCI Express link.
- Sourcing data for most write transactions issued to the PCI Express link. The only exception is certain BFM write procedures that have a four-byte field of write data passed in the call.
- Storing a data structure that contains the sizes of and the values programmed in the BARs of the Endpoint.

A set of procedures is provided to read, write, fill, and check the shared memory from the BFM driver. For details on these procedures, see *BFM Shared Memory Access Procedures*.

- BFM Read/Write Request Functions(**altpcietb_bfm_driver_rp.v**)—These functions provide the basic BFM calls for PCI Express read and write requests. For details on these procedures, refer to *BFM Read and Write Procedures*.
- BFM Configuration Functions(**altpcietb_bfm_driver_rp.v**)—These functions provide the BFM calls to request configuration of the PCI Express link and the Endpoint Configuration Space registers. For details on these procedures and functions, refer to *BFM Configuration Procedures*.
- BFM Log Interface(**altpcietb_bfm_driver_rp.v**)—The BFM log functions provides routines for writing commonly formatted messages to the simulator standard output and optionally to a log file. It also provides controls that stop simulation on errors. For details on these procedures, refer to *BFM Log and Message Procedures*.
- BFM Request Interface(**altpcietb_bfm_driver_rp.v**)—This interface provides the low-level interface between the `altpcietb_bfm_rdwr` and `altpcietb_bfm_configure` procedures or functions and the Root Port RTL Model. This interface stores a write-protected data structure containing the sizes and the values programmed in the BAR registers of the Endpoint, as well as, other critical data used for internal BFM management. You do not need to access these files directly to adapt the testbench to test your Endpoint application.
- Avalon-ST Interfaces (**altpcietb_bfm_vc_intf.v**)—These interface modules handle the Root Port interface model. They take requests from the BFM request interface and generate the required PCI Express transactions. They handle completions received from the PCI Express link and notify the BFM request interface when requests are complete. Additionally, they handle any requests received from the PCI Express link, and store or fetch data from the shared memory before generating the required completions.

Related Information

- **Test Signals**
- **BFM Shared Memory Access Procedures** on page 16-40

BFM Memory Map

The BFM shared memory is configured to be two MBytes. The BFM shared memory is mapped into the first two MBytes of I/O space and also the first two MBytes of memory space. When the Endpoint application generates an I/O or memory transaction in this range, the BFM reads or writes the shared memory.

Configuration Space Bus and Device Numbering

The Root Port interface is assigned to be device number 0 on internal bus number 0. The Endpoint can be assigned to be any device number on any bus number (greater than 0) through the call to procedure `ebfm_cfg_rp_ep`. The specified bus number is assigned to be the secondary bus in the Root Port Configuration Space.

Configuration of Root Port and Endpoint

Before you issue transactions to the Endpoint, you must configure the Root Port and Endpoint Configuration Space registers. To configure these registers, call the procedure `ebfm_cfg_rp_ep`, which is included in **altpcietb_bfm_driver_rp.v**.

The ebfm_cfg_rp_ep executes the following steps to initialize the Configuration Space:

1. Sets the Root Port Configuration Space to enable the Root Port to send transactions on the PCI Express link.
2. Sets the Root Port and Endpoint PCI Express Capability Device Control registers as follows:
   a. Disables Error Reporting in both the Root Port and Endpoint. BFM does not have error handling capability.
   b. Enables Relaxed Ordering in both Root Port and Endpoint.
   c. Enables Extended Tags for the Endpoint, if the Endpoint has that capability.
   d. Disables Phantom Functions, Aux Power PM, and No Snoop in both the Root Port and Endpoint.
   e. Sets the Max Payload Size to what the Endpoint supports because the Root Port supports the maximum payload size.
   f. Sets the Root Port Max Read Request Size to 4 KBytes because the example Endpoint design supports breaking the read into as many completions as necessary.
   g. Sets the Endpoint Max Read Request Size equal to the Max Payload Size because the Root Port does not support breaking the read request into multiple completions.
3. Assigns values to all the Endpoint BAR registers. The BAR addresses are assigned by the algorithm outlined below.
   a. I/O BARs are assigned smallest to largest starting just above the ending address of BFM shared memory in I/O space and continuing as needed throughout a full 32-bit I/O space.
   b. The 32-bit non-prefetchable memory BARs are assigned smallest to largest, starting just above the ending address of BFM shared memory in memory space and continuing as needed throughout a full 32-bit memory space.
   c. Assignment of the 32-bit prefetchable and 64-bit prefetchable memory BARS are based on the value of the addr_map_4GB_limit input to the ebfm_cfg_rp_ep. The default value of the addr_map_4GB_limit is 0.

      If the addr_map_4GB_limit input to the ebfm_cfg_rp_ep is set to 0, then the 32-bit prefetchable memory BARs are assigned largest to smallest, starting at the top of 32-bit memory space and continuing as needed down to the ending address of the last 32-bit non-prefetchable BAR.

      However, if the addr_map_4GB_limit input is set to 1, the address map is limited to 4 GByte, the 32-bit and 64-bit prefetchable memory BARs are assigned largest to smallest, starting at the top of the 32-bit memory space and continuing as needed down to the ending address of the last 32-bit non-prefetchable BAR.
   d. If the addr_map_4GB_limit input to the ebfm_cfg_rp_ep is set to 0, then the 64-bit prefetchable memory BARs are assigned smallest to largest starting at the 4 GByte address assigning memory ascending above the 4 GByte limit throughout the full 64-bit memory space.

      If the addr_map_4 GB_limit input to the ebfm_cfg_rp_ep is set to 1, then the 32-bit and the 64-bit prefetchable memory BARs are assigned largest to smallest starting at the 4 GByte address and assigning memory by descending below the 4 GByte address to addresses memory as needed down to the ending address of the last 32-bit non-prefetchable BAR.

      The above algorithm cannot always assign values to all BARs when there are a few very large (1 GByte or greater) 32-bit BARs. Although assigning addresses to all BARs may be possible, a more complex algorithm would be required to effectively assign these addresses. However, such a

configuration is unlikely to be useful in real systems. If the procedure is unable to assign the BARs, it displays an error message and stops the simulation.

4. Based on the above BAR assignments, the Root Port Configuration Space address windows are assigned to encompass the valid BAR address ranges.
5. The Endpoint PCI control register is set to enable master transactions, memory address decoding, and I/O address decoding.

The `ebfm_cfg_rp_ep` procedure also sets up a `bar_table` data structure in BFM shared memory that lists the sizes and assigned addresses of all Endpoint BARs. This area of BFM shared memory is write-protected, which means any user write accesses to this area cause a fatal simulation error. This data structure is then used by subsequent BFM procedure calls to generate the full PCI Express addresses for read and write requests to particular offsets from a BAR. This procedure allows the testbench code that accesses the Endpoint Application Layer to be written to use offsets from a BAR and not have to keep track of the specific addresses assigned to the BAR. The following table shows how those offsets are used.

Table 16-19: BAR Table Structure

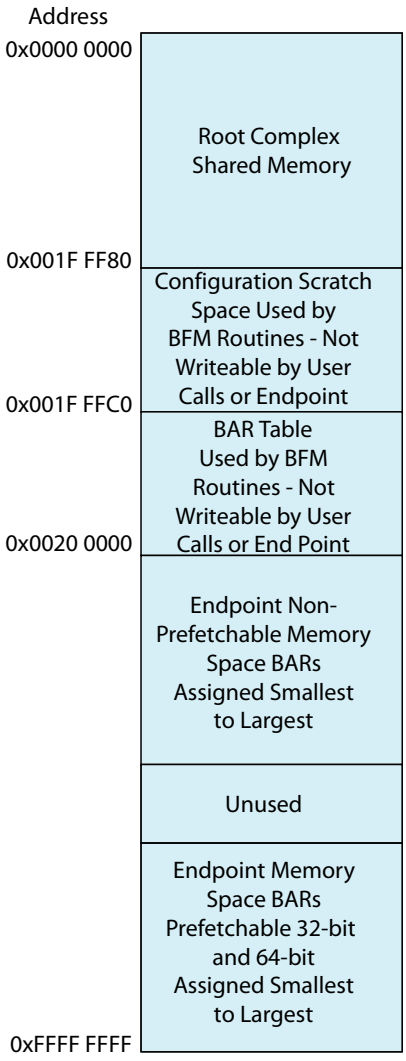
Offset (Bytes)	Description
+0	PCI Express address in BAR0
+4	PCI Express address in BAR1
+8	PCI Express address in BAR2
+12	PCI Express address in BAR3
+16	PCI Express address in BAR4
+20	PCI Express address in BAR5
+24	PCI Express address in Expansion ROM BAR
+28	Reserved
+32	BAR0 read back value after being written with all 1's (used to compute size)
+36	BAR1 read back value after being written with all 1's
+40	BAR2 read back value after being written with all 1's
+44	BAR3 read back value after being written with all 1's
+48	BAR4 read back value after being written with all 1's
+52	BAR5 read back value after being written with all 1's
+56	Expansion ROM BAR read back value after being written with all 1's

Offset (Bytes)	Description
+60	Reserved

The configuration routine does not configure any advanced PCI Express capabilities such as the AER capability.

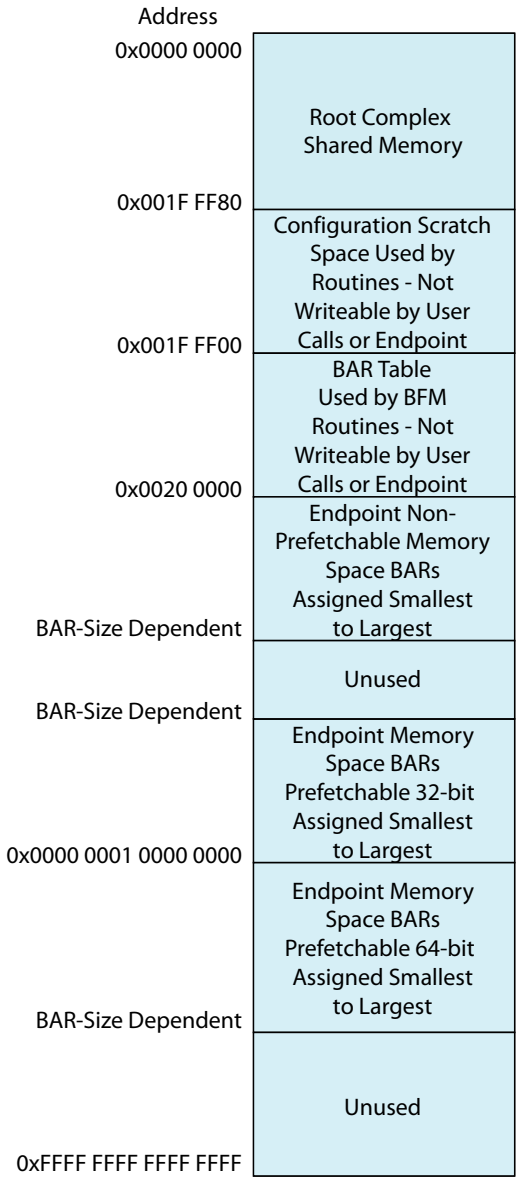
Besides the `ebfm_cfg_rp_ep` procedure in **altpcietb_bfm_driver_rp.v**, routines to read and write Endpoint Configuration Space registers directly are available in the Verilog HDL include file. After the `ebfm_cfg_rp_ep` procedure is run the PCI Express I/O and Memory Spaces have the layout as described in the following three figures. The memory space layout is dependent on the value of the **addr_map_4GB_limit** input parameter. If **addr_map_4GB_limit** is 1 the resulting memory space map is shown in the following figure.

Figure 16-5: Memory Space Layout—4 GByte Limit


Address
0x0000 0000
Root Complex Shared Memory
0x001F FF80
Configuration Scratch Space Used by BFM Routines - Not Writeable by User Calls or Endpoint
0x001F FFC0
BAR Table Used by BFM Routines - Not Writeable by User Calls or End Point
0x0020 0000
Endpoint Non-Prefetchable Memory Space BARs Assigned Smallest to Largest
Unused
Endpoint Memory Space BARs Prefetchable 32-bit and 64-bit Assigned Smallest to Largest
0xFFFF FFFF

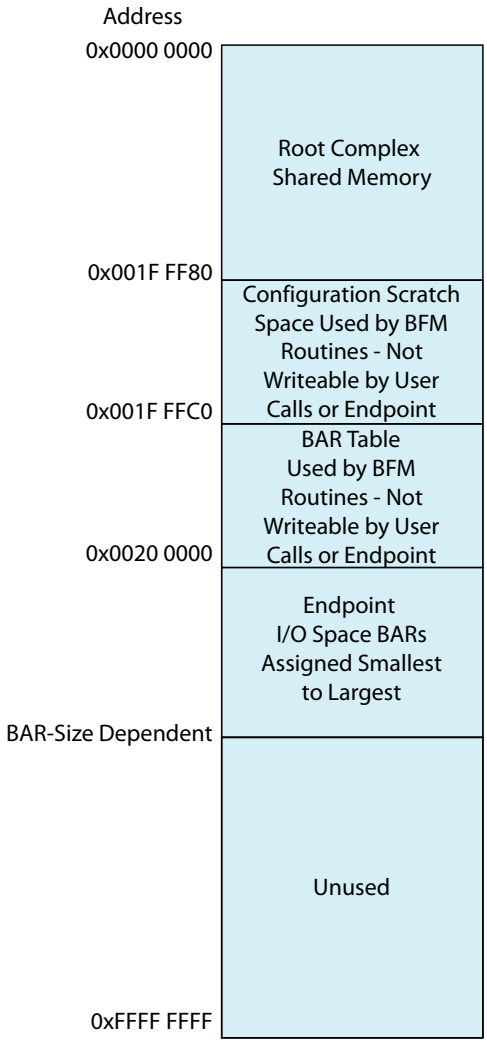
If **addr_map_4GB_limit** is 0, the resulting memory space map is shown in the following figure.

Figure 16-6: Memory Space Layout—No Limit


Address
0x0000 0000
Root Complex Shared Memory
0x001F FF80
Configuration Scratch Space Used by Routines - Not Writeable by User Calls or Endpoint
0x001F FF00
BAR Table Used by BFM Routines - Not Writeable by User Calls or Endpoint
0x0020 0000
Endpoint Non-Prefetchable Memory Space BARs Assigned Smallest to Largest
BAR-Size Dependent
Unused
BAR-Size Dependent
Endpoint Memory Space BARs Prefetchable 32-bit Assigned Smallest to Largest
0x0000 0001 0000 0000
Endpoint Memory Space BARs Prefetchable 64-bit Assigned Smallest to Largest
BAR-Size Dependent
Unused
0xFFFF FFFF FFFF FFFF

The following figure shows the I/O address space.

Figure 16-7: I/O Address Space



Issuing Read and Write Transactions to the Application Layer

Read and write transactions are issued to the Endpoint Application Layer by calling one of the `ebfm_bar` procedures in **altpcietb_bfm_driver_rp.v**. The procedures and functions listed below are available in the

Verilog HDL include file **altpcietb_bfm_driver_rp.v**. The complete list of available procedures and functions is as follows:

- `ebfm_barwr`—writes data from BFM shared memory to an offset from a specific Endpoint BAR. This procedure returns as soon as the request has been passed to the VC interface module for transmission.
- `ebfm_barwr_imm`—writes a maximum of four bytes of immediate data (passed in a procedure call) to an offset from a specific Endpoint BAR. This procedure returns as soon as the request has been passed to the VC interface module for transmission.
- `ebfm_barrd_wait`—reads data from an offset of a specific Endpoint BAR and stores it in BFM shared memory. This procedure blocks waiting for the completion data to be returned before returning control to the caller.
- `ebfm_barrd_nowt`—reads data from an offset of a specific Endpoint BAR and stores it in the BFM shared memory. This procedure returns as soon as the request has been passed to the VC interface module for transmission, allowing subsequent reads to be issued in the interim.

These routines take as parameters a BAR number to access the memory space and the BFM shared memory address of the `bar_table` data structure that was set up by the `ebfm_cfg_rp_ep` procedure. (Refer to *Configuration of Root Port and Endpoint*.) Using these parameters simplifies the BFM test driver routines that access an offset from a specific BAR and eliminates calculating the addresses assigned to the specified BAR.

The Root Port BFM does not support accesses to Endpoint I/O space BARs.

Related Information

Configuration of Root Port and Endpoint on page 16-25

BFM Procedures and Functions

The BFM includes procedures, functions, and tasks to drive Endpoint application testing. It also includes procedures to run the chaining DMA design example.

The BFM read and write procedures read and write data among BFM shared memory, Endpoint BARs, and specified configuration registers. The procedures and functions are available in the Verilog HDL. They are in the include file **altpcietb_bfm_driver.v**. These procedures and functions support issuing memory and configuration transactions on the PCI Express link.

ebfm_barwr Procedure

The `ebfm_barwr` procedure writes a block of data from BFM shared memory to an offset from the specified Endpoint BAR. The length can be longer than the configured `MAXIMUM_PAYLOAD_SIZE`; the procedure breaks the request up into multiple transactions as needed. This routine returns as soon as the last transaction has been accepted by the VC interface module.

Location	altpcietb_bfm_rdwr.v
Syntax	`ebfm_barwr(bar_table, bar_num, pcie_offset, lcladdr, byte_len, tclass)`

Location	altpcietb_bfm_rdwr.v	
Arguments	bar_table	Address of the Endpoint bar_table structure in BFM shared memory. The bar_table structure stores the address assigned to each BAR so that the driver code does not need to be aware of the actual assigned addresses only the application specific offsets from the BAR.
	bar_num	Number of the BAR used with pcie_offset to determine PCI Express address.
	pcie_offset	Address offset from the BAR base.
	lcladdr	BFM shared memory address of the data to be written.
	byte_len	Length, in bytes, of the data written. Can be 1 to the minimum of the bytes remaining in the BAR space or BFM shared memory.
	tclass	Traffic class used for the PCI Express transaction.

ebfm_barwr_imm Procedure

The ebfm_barwr_imm procedure writes up to four bytes of data to an offset from the specified Endpoint BAR.

Location	altpcietb_bfm_driver_rp.v
Syntax	ebfm_barwr_imm(bar_table, bar_num, pcie_offset, imm_data, byte_len, tclass)

Location	altpcietb_bfm_driver_rp.v	
Arguments	bar_table	Address of the Endpoint bar_table structure in BFM shared memory. The bar_table structure stores the address assigned to each BAR so that the driver code does not need to be aware of the actual assigned addresses only the application specific offsets from the BAR.
	bar_num	Number of the BAR used with pcie_offset to determine PCI Express address.
	pcie_offset	Address offset from the BAR base.
	imm_data	Data to be written. In Verilog HDL, this argument is reg [31:0].In both languages, the bits written depend on the length as follows: Length Bits Written • 4: 31 downto 0 • 3: 23 downto 0 • 2: 15 downto 0 • 1: 7 downto 0
	byte_len	Length of the data to be written in bytes. Maximum length is 4 bytes.
	tclass	Traffic class to be used for the PCI Express transaction.

ebfm_barrd_wait Procedure

The ebfm_barrd_wait procedure reads a block of data from the offset of the specified Endpoint BAR and stores it in BFM shared memory. The length can be longer than the configured maximum read request size; the procedure breaks the request up into multiple transactions as needed. This procedure waits until all of the completion data is returned and places it in shared memory.

Location	**altpcietb_bfm_driver_rp.v**
Syntax	ebfm_barrd_wait(bar_table, bar_num, pcie_offset, lcladdr, byte_len, tclass)

Arguments	bar_table	Address of the Endpoint bar_table structure in BFM shared memory. The bar_table structure stores the address assigned to each BAR so that the driver code does not need to be aware of the actual assigned addresses only the application specific offsets from the BAR.
	bar_num	Number of the BAR used with pcie_offset to determine PCI Express address.
	pcie_offset	Address offset from the BAR base.
	lcladdr	BFM shared memory address where the read data is stored.
	byte_len	Length, in bytes, of the data to be read. Can be 1 to the minimum of the bytes remaining in the BAR space or BFM shared memory.
	tclass	Traffic class used for the PCI Express transaction.

ebfm_barrd_nowt Procedure

The ebfm_barrd_nowt procedure reads a block of data from the offset of the specified Endpoint BAR and stores the data in BFM shared memory. The length can be longer than the configured maximum read request size; the procedure breaks the request up into multiple transactions as needed. This routine returns as soon as the last read transaction has been accepted by the VC interface module, allowing subsequent reads to be issued immediately.

Location	altpcietb_b fm_driver_rp.v	
Syntax	ebfm_barrd_nowt(bar_table, bar_num, pcie_offset, lcladdr, byte_len, tclass)	
Arguments	bar_table	Address of the Endpoint bar_table structure in BFM shared memory.
	bar_num	Number of the BAR used with pcie_offset to determine PCI Express address.
	pcie_offset	Address offset from the BAR base.
	lcladdr	BFM shared memory address where the read data is stored.
	byte_len	Length, in bytes, of the data to be read. Can be 1 to the minimum of the bytes remaining in the BAR space or BFM shared memory.
	tclass	Traffic Class to be used for the PCI Express transaction.

ebfm_cfgwr_imm_wait Procedure

The `ebfm_cfgwr_imm_wait` procedure writes up to four bytes of data to the specified configuration register. This procedure waits until the write completion has been returned.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`ebfm_cfgwr_imm_wait(bus_num, dev_num, fnc_num, imm_regb_ad, regb_ln, imm_data, compl_status`	
Arguments	`bus_num`	PCI Express bus number of the target device.
	`dev_num`	PCI Express device number of the target device.
	`fnc_num`	Function number in the target device to be accessed.
	`regb_ad`	Byte-specific address of the register to be written.
	`regb_ln`	Length, in bytes, of the data written. Maximum length is four bytes. The `regb_ln` and the `regb_ad` arguments cannot cross a DWORD boundary.
	`imm_data`	Data to be written. This argument is `reg [31:0]`. The bits written depend on the length: • 4: 31 downto 0 • 3: 23 downto 0 • 2: 15 downto 0 • 1: 7 downto 0
	`compl_status`	This argument is `reg [2:0]`. This argument is the completion status as specified in the PCI Express specification. The following encodings are defined: • 3'b000: SC— Successful completion • 3'b001: UR— Unsupported Request • 3'b010: CRS — Configuration Request Retry Status • 3'b100: CA — Completer Abort

ebfm_cfgwr_imm_nowt Procedure

The `ebfm_cfgwr_imm_nowt` procedure writes up to four bytes of data to the specified configuration register. This procedure returns as soon as the VC interface module accepts the transaction, allowing other writes to be issued in the interim. Use this procedure only when successful completion status is expected.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`ebfm_cfgwr_imm_nowt(bus_num, dev_num, fnc_num, imm_regb_adr, regb_len, imm_data)`	
Arguments	`bus_num`	PCI Express bus number of the target device.
	`dev_num`	PCI Express device number of the target device.
	`fnc_num`	Function number in the target device to be accessed.
	`regb_ad`	Byte-specific address of the register to be written.
	`regb_ln`	Length, in bytes, of the data written. Maximum length is four bytes, The `regb_ln` the `regb_ad` arguments cannot cross a DWORD boundary.
	`imm_data`	Data to be written This argument is `reg [31:0]`. In both languages, the bits written depend on the length. The following encodes are defined. • 4: [31:0] • 3: [23:0] • 2: [15:0] • 1: [7:0]

ebfm_cfgrd_wait Procedure

The `ebfm_cfgrd_wait` procedure reads up to four bytes of data from the specified configuration register and stores the data in BFM shared memory. This procedure waits until the read completion has been returned.

Location	altpcietb_bfm_driver_rp.v
Syntax	`ebfm_cfgrd_wait(bus_num, dev_num, fnc_num, regb_ad, regb_ln, lcladdr, compl_status)`

Location	altpcietb_bfm_driver_rp.v	
Arguments	bus_num	PCI Express bus number of the target device.
	dev_num	PCI Express device number of the target device.
	fnc_num	Function number in the target device to be accessed.
	regb_ad	Byte-specific address of the register to be written.
	regb_ln	Length, in bytes, of the data read. Maximum length is four bytes. The regb_ln and the regb_ad arguments cannot cross a DWORD boundary.
	lcladdr	BFM shared memory address of where the read data should be placed.
	compl_status	Completion status for the configuration transaction. This argument is reg [2:0]. In both languages, this is the completion status as specified in the PCI Express specification. The following encodings are defined. • 3'b000: SC— Successful completion • 3'b001: UR— Unsupported Request • 3'b010: CRS — Configuration Request Retry Status • 3'b100: CA — Completer Abort

ebfm_cfgrd_nowt Procedure

The ebfm_cfgrd_nowt procedure reads up to four bytes of data from the specified configuration register and stores the data in the BFM shared memory. This procedure returns as soon as the VC interface module has accepted the transaction, allowing other reads to be issued in the interim. Use this procedure only when successful completion status is expected and a subsequent read or write with a wait can be used to guarantee the completion of this operation.

Location	altpcietb_bfm_driver_rp.v
Syntax	ebfm_cfgrd_nowt(bus_num, dev_num, fnc_num, regb_ad, regb_ln, lcladdr)

Location	altpcietb_bfm_driver_rp.v	
Arguments	bus_num	PCI Express bus number of the target device.
	dev_num	PCI Express device number of the target device.
	fnc_num	Function number in the target device to be accessed.
	regb_ad	Byte-specific address of the register to be written.
	regb_ln	Length, in bytes, of the data written. Maximum length is four bytes. The regb_ln and regb_ad arguments cannot cross a DWORD boundary.
	lcladdr	BFM shared memory address where the read data should be placed.

BFM Configuration Procedures

The BFM configuration procedures are available in **altpcietb_bfm_driver_rp.v**. These procedures support configuration of the Root Port and Endpoint Configuration Space registers.

All Verilog HDL arguments are type `integer` and are input-only unless specified otherwise.

ebfm_cfg_rp_ep Procedure

The `ebfm_cfg_rp_ep` procedure configures the Root Port and Endpoint Configuration Space registers for operation.

Location	altpcietb_bfm_driver_rp.v
Syntax	`ebfm_cfg_rp_ep(bar_table, ep_bus_num, ep_dev_num, rp_max_rd_req_size, display_ep_config, addr_map_4GB_limit)`

Location	altpcietb_bfm_driver_rp.v	
Arguments	bar_table	Address of the Endpoint bar_table structure in BFM shared memory. This routine populates the bar_table structure. The bar_table structure stores the size of each BAR and the address values assigned to each BAR. The address of the bar_table structure is passed to all subsequent read and write procedure calls that access an offset from a particular BAR.
	ep_bus_num	PCI Express bus number of the target device. This number can be any value greater than 0. The Root Port uses this as its secondary bus number.
	ep_dev_num	PCI Express device number of the target device. This number can be any value. The Endpoint is automatically assigned this value when it receives its first configuration transaction.
	rp_max_rd_req_size	Maximum read request size in bytes for reads issued by the Root Port. This parameter must be set to the maximum value supported by the Endpoint Application Layer. If the Application Layer only supports reads of the MAXIMUM_PAYLOAD_SIZE, then this can be set to 0 and the read request size will be set to the maximum payload size. Valid values for this argument are 0, 128, 256, 512, 1,024, 2,048 and 4,096.
	display_ep_config	When set to 1 many of the Endpoint Configuration Space registers are displayed after they have been initialized, causing some additional reads of registers that are not normally accessed during the configuration process such as the Device ID and Vendor ID.
	addr_map_4GB_limit	When set to 1 the address map of the simulation system will be limited to 4 GBytes. Any 64-bit BARs will be assigned below the 4 GByte limit.

ebfm_cfg_decode_bar Procedure

The ebfm_cfg_decode_bar procedure analyzes the information in the BAR table for the specified BAR and returns details about the BAR attributes.

Location	altpcietb_bfm_driver_rp.v
Syntax	ebfm_cfg_decode_bar(bar_table, bar_num, log2_size, is_mem, is_pref, is_64b)

Location	altpcietb_bfm_driver_rp.v	
Arguments	bar_table	Address of the Endpoint bar_table structure in BFM shared memory.
	bar_num	BAR number to analyze.
	log2_size	This argument is set by the procedure to the log base 2 of the size of the BAR. If the BAR is not enabled, this argument will be set to 0.
	is_mem	The procedure sets this argument to indicate if the BAR is a memory space BAR (1) or I/O Space BAR (0).
	is_pref	The procedure sets this argument to indicate if the BAR is a prefetchable BAR (1) or non-prefetchable BAR (0).
	is_64b	The procedure sets this argument to indicate if the BAR is a 64-bit BAR (1) or 32-bit BAR (0). This is set to 1 only for the lower numbered BAR of the pair.

BFM Shared Memory Access Procedures

The BFM shared memory access procedures and functions are in the Verilog HDL include file **altpcietb_bfm_driver.v**. These procedures and functions support accessing the BFM shared memory.

Shared Memory Constants

The following constants are defined in **altpcietb_bfm_driver.v**. They select a data pattern in the `shmem_fill` and `shmem_chk_ok` routines. These shared memory constants are all Verilog HDL type `integer`.

Table 16-20: Constants: Verilog HDL Type INTEGER

Constant	Description
SHMEM_FILL_ZEROS	Specifies a data pattern of all zeros
SHMEM_FILL_BYTE_INC	Specifies a data pattern of incrementing 8-bit bytes (0x00, 0x01, 0x02, etc.)
SHMEM_FILL_WORD_INC	Specifies a data pattern of incrementing 16-bit words (0x0000, 0x0001, 0x0002, etc.)
SHMEM_FILL_DWORD_INC	Specifies a data pattern of incrementing 32-bit dwords (0x00000000, 0x00000001, 0x00000002, etc.)

Constant	Description
SHMEM_FILL_QWORD_INC	Specifies a data pattern of incrementing 64-bit qwords (0x0000000000000000, 0x0000000000000001, 0x0000000000000002, etc.)
SHMEM_FILL_ONE	Specifies a data pattern of all ones

shmem_write

The shmem_write procedure writes data to the BFM shared memory.

Location	altpcietb_bfm_driver_rp.v	
Syntax	shmem_write(addr, data, leng)	
Arguments	addr	BFM shared memory starting address for writing data
	data	Data to write to BFM shared memory. This parameter is implemented as a 64-bit vector. leng is 1–8 bytes. Bits 7 downto 0 are written to the location specified by addr; bits 15 downto 8 are written to the addr+1 location, etc.
	length	Length, in bytes, of data written

shmem_read Function

The shmem_read function reads data to the BFM shared memory.

Location	altpcietb_bfm_driver_rp.v	
Syntax	data:= shmem_read(addr, leng)	
Arguments	addr	BFM shared memory starting address for reading data
	leng	Length, in bytes, of data read
Return	data	Data read from BFM shared memory. This parameter is implemented as a 64-bit vector. leng is 1- 8 bytes. If leng is less than 8 bytes, only the corresponding least significant bits of the returned data are valid. Bits 7 downto 0 are read from the location specified by addr; bits 15 downto 8 are read from the addr+1 location, etc.

shmem_display Verilog HDL Function

The `shmem_display` Verilog HDL function displays a block of data from the BFM shared memory.

Location	altpcietb_bfm_driver_rp.v	
Syntax	Verilog HDL: `dummy_return:=shmem_display(addr, leng, word_size, flag_addr, msg_type);`	
Arguments	`addr`	BFM shared memory starting address for displaying data.
	`leng`	Length, in bytes, of data to display.
	`word_size`	Size of the words to display. Groups individual bytes into words. Valid values are 1, 2, 4, and 8.
	`flag_addr`	Adds a <== flag to the end of the display line containing this address. Useful for marking specific data. Set to a value greater than 2**21 (size of BFM shared memory) to suppress the flag.
	`msg_type`	Specifies the message type to be displayed at the beginning of each line. See "BFM Log and Message Procedures" on page 18–37 for more information about message types. Set to one of the constants defined in Table 18–36 on page 18–41.

shmem_fill Procedure

The `shmem_fill` procedure fills a block of BFM shared memory with a specified data pattern.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`shmem_fill(addr, mode, leng, init)`	
Arguments	`addr`	BFM shared memory starting address for filling data.
	`mode`	Data pattern used for filling the data. Should be one of the constants defined in section *Shared Memory Constants*.
	`leng`	Length, in bytes, of data to fill. If the length is not a multiple of the incrementing data pattern width, then the last data pattern is truncated to fit.
	`init`	Initial data value used for incrementing data pattern modes. This argument is `reg [63:0]`. The necessary least significant bits are used for the data patterns that are smaller than 64 bits.

Related Information

Shared Memory Constants on page 16-40

shmem_chk_ok Function

The `shmem_chk_ok` function checks a block of BFM shared memory against a specified data pattern.

Location	altpcietb_bfm_shmem.v	
Syntax	`result:= shmem_chk_ok(addr, mode, leng, init, display_error)`	
Arguments	`addr`	BFM shared memory starting address for checking data.
	`mode`	Data pattern used for checking the data. Should be one of the constants defined in section "Shared Memory Constants" on page 18–35.
	`leng`	Length, in bytes, of data to check.
	`init`	This argument is `reg [63:0]`.The necessary least significant bits are used for the data patterns that are smaller than 64-bits.
	`display_error`	When set to 1, this argument displays the mis-comparing data on the simulator standard output.
Return	`Result`	Result is 1-bit. • 1'b1 — Data patterns compared successfully • 1'b0 — Data patterns did not compare successfully

BFM Log and Message Procedures

The following procedures and functions are available in the Verilog HDL include file **altpcietb_bfm_driver_rp.v.**

These procedures provide support for displaying messages in a common format, suppressing informational messages, and stopping simulation on specific message types.

The following constants define the type of message and their values determine whether a message is displayed or simulation is stopped after a specific message. Each displayed message has a specific prefix, based on the message type in the following table.

You can suppress the display of certain message types. The default values determining whether a message type is displayed are defined in the following table. To change the default message display, modify the display default value with a procedure call to `ebfm_log_set_suppressed_msg_mask`.

Certain message types also stop simulation after the message is displayed. The following table shows the default value determining whether a message type stops simulation. You can specify whether simulation stops for particular messages with the procedure `ebfm_log_set_stop_on_msg_mask`.

All of these log message constants type `integer`.

Table 16-21: Log Messages

Constant (Message Type)	Description	Mask Bit No	Display by Default	Simulation Stops by Default	Message Prefix
EBFM_MSG_DEBUG	Specifies debug messages.	0	No	No	DEBUG:
EBFM_MSG_INFO	Specifies informational messages, such as configuration register values, starting and ending of tests.	1	Yes	No	INFO:
EBFM_MSG_WARNING	Specifies warning messages, such as tests being skipped due to the specific configuration.	2	Yes	No	WARNING:
EBFM_MSG_ERROR_INFO	Specifies additional information for an error. Use this message to display preliminary information before an error message that stops simulation.	3	Yes	No	ERROR:
EBFM_MSG_ERROR_CONTINUE	Specifies a recoverable error that allows simulation to continue. Use this error for data miscompares.	4	Yes	No	ERROR:
EBFM_MSG_ERROR_FATAL	Specifies an error that stops simulation because the error leaves the testbench in a state where further simulation is not possible.	N/A	Yes Cannot suppress	Yes Cannot suppress	FATAL:

Constant (Message Type)	Description	Mask Bit No	Display by Default	Simulation Stops by Default	Message Prefix
EBFM_MSG_ERROR_FATAL_TB_ERR	Used for BFM test driver or Root Port BFM fatal errors. Specifies an error that stops simulation because the error leaves the testbench in a state where further simulation is not possible. Use this error message for errors that occur due to a problem in the BFM test driver module or the Root Port BFM, that are not caused by the Endpoint Application Layer being tested.	N/A	Y Cannot suppress	Y Cannot suppress	FATAL:

ebfm_display Verilog HDL Function

The ebfm_display procedure or function displays a message of the specified type to the simulation standard output and also the log file if ebfm_log_open is called.

A message can be suppressed, simulation can be stopped or both based on the default settings of the message type and the value of the bit mask when each of the procedures listed below is called. You can call one or both of these procedures based on what messages you want displayed and whether or not you want simulation to stop for specific messages.

- When eb fm_log_set_suppressed_msg_mask is called, the display of the message might be suppressed based on the value of the bit mask.
- When ebfm_log_set_stop_on_msg_mask is called, the simulation can be stopped after the message is displayed, based on the value of the bit mask.

Location	altpcietb_bfm_driver_rp.v	
Syntax	Verilog HDL: dummy_return:=ebfm_display(msg_type, message);	
Argument	msg_type	Message type for the message. Should be one of the constants defined in Table 18–36 on page 18–41.
	message	The message string is limited to a maximum of 100 characters. Also, because Verilog HDL does not allow variable length strings, this routine strips off leading characters of 8'h00 before displaying the message.
Return	always 0	Applies only to the Verilog HDL routine.

ebfm_log_stop_sim Verilog HDL Function

The `ebfm_log_stop_sim` procedure stops the simulation.

Location	altpcietb_bfm_driver_rp.v	
Syntax	Verilog VHDL: `return:=ebfm_log_stop_sim(success);`	
Argument	`success`	When set to a 1, this process stops the simulation with a message indicating successful completion. The message is prefixed with `SUCCESS:`. Otherwise, this process stops the simulation with a message indicating unsuccessful completion. The message is prefixed with `FAILURE:`.
Return	Always 0	This value applies only to the Verilog HDL function.

ebfm_log_set_suppressed_msg_mask #Verilog HDL Function

The `ebfm_log_set_suppressed_msg_mask` procedure controls which message types are suppressed.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`bfm_log_set_suppressed_msg_mask (msg_mask)`	
Argument	`msg_mask`	This argument is `reg [EBFM_MSG_ERROR_CONTINUE: EBFM_MSG_DEBUG].` A 1 in a specific bit position of the `msg_mask` causes messages of the type corresponding to the bit position to be suppressed.

ebfm_log_set_stop_on_msg_mask Verilog HDL Function

The `ebfm_log_set_stop_on_msg_mask` procedure controls which message types stop simulation. This procedure alters the default behavior of the simulation when errors occur as described in the *BFM Log and Message Procedures*.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`ebfm_log_set_stop_on_msg_mask (msg_mask)`	
Argument	`msg_mask`	This argument is `reg [EBFM_MSG_ERROR_CONTINUE:EBFM_MSG_DEBUG].` A 1 in a specific bit position of the `msg_mask` causes messages of the type corresponding to the bit position to stop the simulation after the message is displayed.

Related Information

BFM Log and Message Procedures on page 16-43

ebfm_log_open Verilog HDL Function

The `ebfm_log_open` procedure opens a log file of the specified name. All displayed messages are called by `ebfm_display` and are written to this log file as simulator standard output.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`ebfm_log_open (fn)`	
Argument	`fn`	This argument is type `string` and provides the file name of log file to be opened.

ebfm_log_close Verilog HDL Function

The `ebfm_log_close` procedure closes the log file opened by a previous call to `ebfm_log_open`.

Location	altpcietb_bfm_driver_rp.v
Syntax	`ebfm_log_close`
Argument	NONE

Verilog HDL Formatting Functions

The Verilog HDL Formatting procedures and functions are available in the **altpcietb_bfm_driver_rp.v**. The formatting functions are only used by Verilog HDL. All these functions take one argument of a specified length and return a vector of a specified length.

himage1

This function creates a one-digit hexadecimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`string:= himage(vec)`	
Argument	`vec`	Input data type `reg` with a `range` of 3:0.
Return range	`string`	Returns a 1-digit hexadecimal representation of the input argument. Return data is type `reg` with a `range` of 8:1

himage2

This function creates a two-digit hexadecimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`string:= himage(vec)`	
Argument range	`vec`	Input data type `reg` with a `range` of 7:0.
Return range	`string`	Returns a 2-digit hexadecimal presentation of the input argument, padded with leading 0s, if they are needed. Return data is type `reg` with a `range` of 16:1

himage4

This function creates a four-digit hexadecimal string representation of the input argument can be concatenated into a larger message string and passed to `ebfm_display`.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`string:= himage(vec)`	
Argument range	`vec`	Input data type `reg` with a `range` of 15:0.
Return range	Returns a four-digit hexadecimal representation of the input argument, padded with leading 0s, if they are needed. Return data is type `reg` with a `range` of 32:1.	

himage8

This function creates an 8-digit hexadecimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`string:= himage(vec)`	
Argument range	`vec`	Input data type reg with a range of 31:0.
Return range	`string`	Returns an 8-digit hexadecimal representation of the input argument, padded with leading 0s, if they are needed. Return data is type `reg` with a `range` of 64:1.

himage16

This function creates a 16-digit hexadecimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`string:= himage(vec)`	
Argument range	`vec`	Input data type reg with a range of 63:0.
Return range	`string`	Returns a 16-digit hexadecimal representation of the input argument, padded with leading 0s, if they are needed. Return data is type `reg` with a `range` of 128:1.

dimage1

This function creates a one-digit decimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`string:= dimage(vec)`	
Argument range	`vec`	Input data type `reg` with a `range` of 31:0.
Return range	`string`	Returns a 1-digit decimal representation of the input argument that is padded with leading 0s if necessary. Return data is type `reg` with a `range` of 8:1. Returns the letter *U* if the value cannot be represented.

dimage2

This function creates a two-digit decimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`string:= dimage(vec)`	
Argument range	`vec`	Input data type `reg` with a `range` of 31:0.
Return range	`string`	Returns a 2-digit decimal representation of the input argument that is padded with leading 0s if necessary. Return data is type `reg` with a `range` of 16:1. Returns the letter *U* if the value cannot be represented.

dimage3

This function creates a three-digit decimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location	`altpcietb_bfm_driver_rp.v`	
Syntax	`string:= dimage(vec)`	
Argument range	`vec`	Input data type `reg` with a `range` of 31:0.
Return range	string	Returns a 3-digit decimal representation of the input argument that is padded with leading 0s if necessary. Return data is type `reg` with a `range` of 24:1. Returns the letter *U* if the value cannot be represented.

dimage4

This function creates a four-digit decimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location	`altpcietb_bfm_driver_rp.v`	
Syntax	`string:= dimage(vec)`	
Argument range	`vec`	Input data type `reg` with a `range` of 31:0.
Return range	`string`	Returns a 4-digit decimal representation of the input argument that is padded with leading 0s if necessary. Return data is type `reg` with a `range` of 32:1. Returns the letter *U* if the value cannot be represented.

dimage5

This function creates a five-digit decimal string representation of the input argument that can be concatenated into a larger message string and passed to `ebfm_display`.

Location	`altpcietb_bfm_driver_rp.v`	
Syntax	`string:= dimage(vec)`	
Argument range	`vec`	Input data type `reg` with a `range` of 31:0.

Location	altpcietb_bfm_driver_rp.v	
Return range	string	Returns a 5-digit decimal representation of the input argument that is padded with leading 0s if necessary. Return data is type reg with a range of 40:1. Returns the letter *U* if the value cannot be represented.

dimage6

This function creates a six-digit decimal string representation of the input argument that can be concatenated into a larger message string and passed to ebfm_display.

Location	altpcietb_bfm_log.v	
Syntax	string:= dimage(vec)	
Argument range	vec	Input data type reg with a range of 31:0.
Return range	string	Returns a 6-digit decimal representation of the input argument that is padded with leading 0s if necessary. Return data is type reg with a range of 48:1. Returns the letter *U* if the value cannot be represented.

dimage7

This function creates a seven-digit decimal string representation of the input argument that can be concatenated into a larger message string and passed to ebfm_display.

Location	altpcietb_bfm_log.v	
Syntax	string:= dimage(vec)	
Argument range	vec	Input data type reg with a range of 31:0.
Return range	string	Returns a 7-digit decimal representation of the input argument that is padded with leading 0s if necessary. Return data is type reg with a range of 56:1. Returns the letter <*U*> if the value cannot be represented.

Procedures and Functions Specific to the Chaining DMA Design Example

The procedures specific to the chaining DMA design example are in the Verilog HDL module file **altpcietb_bfm_driver_rp.v**.

chained_dma_test Procedure

The `chained_dma_test` procedure is the top-level procedure that runs the chaining DMA read and the chaining DMA write

Location	altpcietb_bfm_driver_rp.v	
Syntax	`chained_dma_test (bar_table, bar_num, direction, use_msi, use_eplast)`	
Arguments	`bar_table`	Address of the Endpoint `bar_table` structure in BFM shared memory.
	`bar_num`	BAR number to analyze.
	`direction`	When 0 the direction is read. When 1 the direction is write.
	`Use_msi`	When set, the Root Port uses native PCI Express MSI to detect the DMA completion.
	`Use_eplast`	When set, the Root Port uses BFM shared memory polling to detect the DMA completion.

dma_rd_test Procedure

Use the `dma_rd_test` procedure for DMA reads from the Endpoint memory to the BFM shared memory.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`dma_rd_test (bar_table, bar_num, use_msi, use_eplast)`	
Arguments	`bar_table`	Address of the Endpoint `bar_table` structure in BFM shared memory.
	`bar_num`	BAR number to analyze.
	`Use_msi`	When set, the Root Port uses native PCI express MSI to detect the DMA completion.
	`Use_eplast`	When set, the Root Port uses BFM shared memory polling to detect the DMA completion.

dma_wr_test Procedure

Use the `dma_wr_test` procedure for DMA writes from the BFM shared memory to the Endpoint memory.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`dma_wr_test (bar_table, bar_num, use_msi, use_eplast)`	
Arguments	`bar_table`	Address of the Endpoint `bar_table` structure in BFM shared memory.
	`bar_num`	BAR number to analyze.
	`Use_msi`	When set, the Root Port uses native PCI Express MSI to detect the DMA completion.
	`Use_eplast`	When set, the Root Port uses BFM shared memory polling to detect the DMA completion.

dma_set_rd_desc_data Procedure

Use the `dma_set_rd_desc_data` procedure to configure the BFM shared memory for the DMA read.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`dma_set_rd_desc_data (bar_table, bar_num)`	
`Arguments`	`bar_table`	Address of the Endpoint `bar_table` structure in BFM shared memory.
	`bar_num`	BAR number to analyze.

dma_set_wr_desc_data Procedure

Use the `dma_set_wr_desc_data` procedure to configure the BFM shared memory for the DMA write.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`dma_set_wr_desc_data_header (bar_table, bar_num)`	
Arguments	`bar_table`	Address of the Endpoint `bar_table` structure in BFM shared memory.
	`bar_num`	BAR number to analyze.

dma_set_header Procedure

Use the `dma_set_header` procedure to configure the DMA descriptor table for DMA read or DMA write.

Location	altpcietb_bfm_driver_rp.v	
Syntax	`dma_set_header (bar_table, bar_num, Descriptor_size, direction, Use_msi, Use_eplast, Bdt_msb, Bdt_lab, Msi_number, Msi_traffic_class, Multi_message_enable)`	
Arguments	`bar_table`	Address of the Endpoint `bar_table` structure in BFM shared memory.
	`bar_num`	BAR number to analyze.
	`Descriptor_size`	Number of descriptor.
	`direction`	When 0 the direction is read. When 1 the direction is write.
	`Use_msi`	When set, the Root Port uses native PCI Express MSI to detect the DMA completion.
	`Use_eplast`	When set, the Root Port uses BFM shared memory polling to detect the DMA completion.
	`Bdt_msb`	BFM shared memory upper address value.
	`Bdt_lsb`	BFM shared memory lower address value.
	`Msi_number`	When `use_msi` is set, specifies the number of the MSI which is set by the `dma_set_msi` procedure.
	`Msi_traffic_class`	When `use_msi` is set, specifies the MSI traffic class which is set by the `dma_set_msi` procedure.
	`Multi_message_enable`	When `use_msi` is set, specifies the MSI traffic class which is set by the `dma_set_msi` procedure.

rc_mempoll Procedure

Use the `rc_mempoll` procedure to poll a given dword in a given BFM shared memory location.

Location	altpcietb_bfm_driver_rp.v
Syntax	`rc_mempoll (rc_addr, rc_data, rc_mask)`

Location	altpcietb_bfm_driver_rp.v	
Arguments	rc_addr	Address of the BFM shared memory that is being polled.
	rc_data	Expected data value of the that is being polled.
	rc_mask	Mask that is logically ANDed with the shared memory data before it is compared with rc_data.

msi_poll Procedure

The msi_poll procedure tracks MSI completion from the Endpoint.

Location	altpcietb_bfm_driver_rp.v	
Syntax	msi_poll(max_number_of_msi,msi_address,msi_expected_dmawr,msi_expected_dmard,dma_write,dma_read)	
Arguments	max_number_of_msi	Specifies the number of MSI interrupts to wait for.
	msi_address	The shared memory location to which the MSI messages will be written.
	msi_expected_dmawr	When dma_write is set, this specifies the expected MSI data value for the write DMA interrupts which is set by the dma_set_msi procedure.
	msi_expected_dmard	When the dma_read is set, this specifies the expected MSI data value for the read DMA interrupts which is set by the dma_set_msi procedure.
	Dma_write	When set, poll for MSI from the DMA write module.
	Dma_read	When set, poll for MSI from the DMA read module.

dma_set_msi Procedure

The dma_set_msi procedure sets PCI Express native MSI for the DMA read or the DMA write.

Location	altpcietb_bfm_driver_rp.v
Syntax	dma_set_msi(bar_table, bar_num, bus_num, dev_num, fun_num, direction, msi_address, msi_data, msi_number, msi_traffic_class, multi_message_enable, msi_expected)

Location	altpcietb_bfm_driver_rp.v	
Arguments	bar_table	Address of the Endpoint bar_table structure in BFM shared memory.
	bar_num	BAR number to analyze.
	Bus_num	Set configuration bus number.
	dev_num	Set configuration device number.
	Fun_num	Set configuration function number.
	Direction	When 0 the direction is read. When 1 the direction is write.
	msi_address	Specifies the location in shared memory where the MSI message data will be stored.
	msi_data	The 16-bit message data that will be stored when an MSI message is sent. The lower bits of the message data will be modified with the message number as per the PCI specifications.
	Msi_number	Returns the MSI number to be used for these interrupts.
	Msi_traffic_class	Returns the MSI traffic class value.
	Multi_message_enable	Returns the MSI multi message enable status.
	msi_expected	Returns the expected MSI data value, which is msi_data modified by the msi_number chosen.

find_mem_bar Procedure

The find_mem_bar procedure locates a BAR which satisfies a given memory space requirement.

Location	altpcietb_bfm_driver_rp.v
Syntax	Find_mem_bar(bar_table,allowed_bars,min_log2_size, sel_bar)

Location	altpcietb_bfm_driver_rp.v	
Arguments	bar_table	Address of the Endpoint bar_table structure in BFM shared memory
	allowed_bars	One hot 6 bits BAR selection
	min_log2_size	Number of bit required for the specified address space
	sel_bar	BAR number to use

dma_set_rclast Procedure

The dma_set_rclast procedure starts the DMA operation by writing to the Endpoint DMA register the value of the last descriptor to process (RCLast).

Location	altpcietb_bfm_driver_rp.v	
Syntax	Dma_set_rclast(bar_table, setup_bar, dt_direction, dt_rclast)	
Arguments	bar_table	Address of the Endpoint bar_table structure in BFM shared memory
	setup_bar	BAR number to use
	dt_direction	When 0 read, When 1 write
	dt_rclast	Last descriptor number

ebfm_display_verb Procedure

The ebfm_display_verb procedure calls the procedure ebfm_display when the global variable DISPLAY_ALL is set to 1.

Location	altpcietb_bfm_driver_chaining.v	
Syntax	ebfm_display_verb(msg_type, message)	
Arguments	msg_type	Message type for the message. Should be one of the constants defined in *BFM Log and Message Procedures*.
	message	The message string is limited to a maximum of 100 characters. Also, because Verilog HDL does not allow variable length strings, this routine strips off leading characters of 8'h00 before displaying the message.

Related Information

BFM Log and Message Procedures on page 16-43

Debugging Simulations

You can modify the following default testbench parameter settings to facilitate debugging.

- For Gen1 and Gen2 variants, you can disable 8B/10B encoding and decoding by setting `test_in[2] = 1` in **altpcietb_bfm_top_rp.v**.
- You can view the most important PIPE interface signals, `txdata`, `txdatak`, `rxdata`, and `txdatak`at the following level of the design hierarchy: `altpcie_<dev>_hip_pipen1b|twentynm_hssi_<gen>_<lanes>_pcie_hip_rbc`.

Debugging 17

2014.12.15

UG-01110_avst

Subscribe Send Feedback

As you bring up your PCI Express system, you may face a number of issues related to FPGA configuration, link training, BIOS enumeration, data transfer, and so on. This chapter suggests some strategies to resolve the common issues that occur during hardware bring-up.

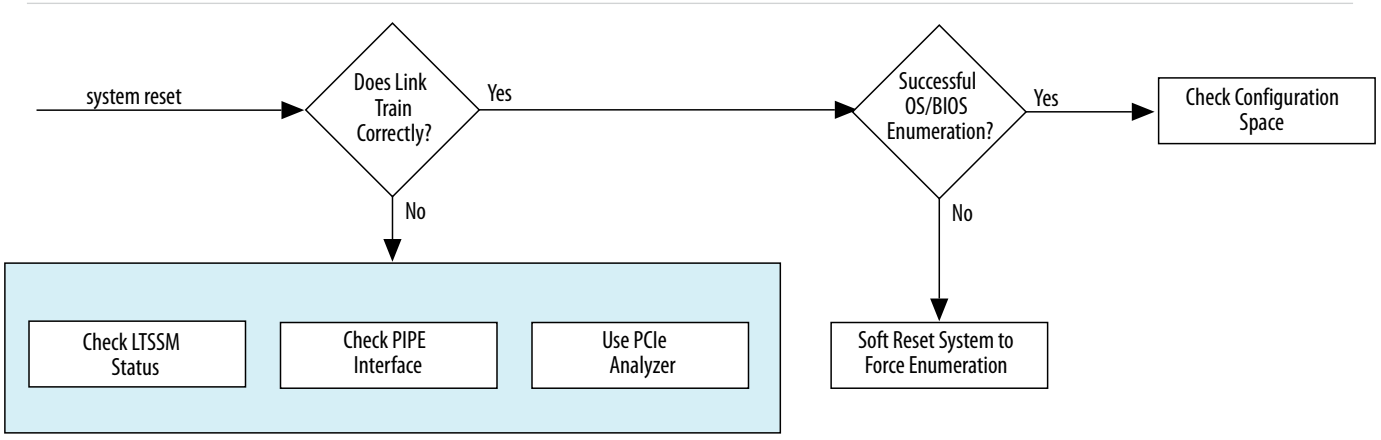
Hardware Bring-Up Issues

Typically, PCI Express hardware bring-up involves the following steps:

1. System reset
2. Link training
3. BIOS enumeration

The following sections, describe how to debug the hardware bring-up flow. Altera recommends a systematic approach to diagnosing bring-up issues as illustrated in the following figure.

Figure 17-1: Debugging Link Training Issues



Link Training

The Physical Layer automatically performs link training and initialization without software intervention. This is a well-defined process to configure and initialize the device's Physical Layer and link so that PCIe

© 2014 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

packets can be transmitted. If you encounter link training issues, viewing the actual data in hardware should help you determine the root cause. You can use the following tools to provide hardware visibility:

- SignalTap II Embedded Logic Analyzer
- Third-party PCIe analyzer

You can use SignalTap II Embedded Logic Analyzer to diagnose the LTSSM state transitions that are occurring on the PIPE interface. The `ltssmstate[4:0]` bus encodes the status of LTSSM. The LTSSM state machine reflects the Physical Layer's progress through the link training process. For a complete description of the states these signals encode, refer to *Status, Link Training and Reset Signals*. When link training completes successfully and the link is up, the LTSSM should remain stable in the L0 state. When link issues occur, you can monitor `ltssmstate[4:0]` to determine the cause.

Related Information

Reset, Status, and Link Training Signals on page 4-24

Debugging Link Failure in L0 Due To Deassertion of tx_st_ready

There are many reasons that link may stop transmitting data. The following table lists some possible causes.

Table 17-1: Link Hangs in L0

Possible Causes	Symptoms and Root Causes	Workarounds and Solutions
Avalon-ST signaling violates Avalon-ST protocol	Avalon-ST protocol violations include the following errors: • More than one `tx_st_sop` per `tx_st_eop`. • Two or more `tx_st_eop's` without a corresponding `tx_st_sop`. • `rx_st_valid` is not asserted with `tx_st_sop` or `tx_st_eop`. These errors are applicable to both simulation and hardware.	Add logic to detect situations where `tx_st_ready` remains deasserted for more than 100 cycles. Set post-triggering conditions to check for the Avalon-ST signalling of last two TLPs to verify correct `tx_st_sop` and `tx_st_eop` signalling.
Incorrect payload size	Determine if the length field of the last TLP transmitted by End Point is greater than the InitFC credit advertised by the link partner. For simulation, refer to the log file and simulation dump. For hardware, use a third-party logic analyzer trace to capture PCIe transactions.	If the payload is greater than the initFC credit advertised, you must either increase the InitFC of the posted request to be greater than the **max payload size** or reduce the payload size of the requested TLP to be less than the InitFC value.

Possible Causes	Symptoms and Root Causes	Workarounds and Solutions
Flow control credit overflows	Determine if the credit field associated with the current TLP type in the `tx_cred` bus is less than the requested credit value. When insufficient credits are available, the core waits for the link partner to release the correct credit type. Sufficient credits may be unavailable if the link partner increments credits more than expected, creating a situation where the Cyclone V Hard IP for PCI Express IP Core credit calculation is out-of-sink with its link partner.	Add logic to detect conditions where the `tx_st_ready` signal remains deasserted for more than 100 cycles. Set post-triggering conditions to check the value of the `tx_cred*` and `tx_st_*` interfaces. Add a FIFO status signal to determine if the TXFIFO is full.
Malformed TLP is transmitted	Refer to the error log file to find the last good packet transmitted on the link. Correlate this packet with TLP sent on Avalon-ST interface. Determine if the last TLP sent has any of the following errors: • The actual payload sent does not match the length field. • The byte enable signals violate rules for byte enables as specified in the *Avalon Interface Specifications*. • The format and type fields are incorrectly specified. • TD field is asserted, indicating the presence of a TLP digest (ECRC), but the ECRC dword is not present at the end of TLP. • The payload crosses a 4KByte boundary.	Revise the Application Layer logic to correct the error condition.

Possible Causes	Symptoms and Root Causes	Workarounds and Solutions
Insufficient Posted credits released by Root Port	If a Memory Write TLP is transmitted with a payload greater than the **maximum payload size**, the Root Port may release an incorrect posted data credit to the Endpoint in simulation. As a result, the Endpoint does not have enough credits to send additional Memory Write Requests.	Make sure Application Layer sends Memory Write Requests with a payload less than or equal the value specified by the **maximum payload size**.
Missing completion packets or dropped packets	The RX Completion TLP might cause the RX FIFO to overflow. Make sure that the total outstanding read data of all pending Memory Read Requests is smaller than the allocated completion credits in RX buffer.	You must ensure that the data for all outstanding read requests does not exceed the completion credits in the RX buffer.

Related Information

- **PIPE Interface Signals** on page 4-50
- **Avalon Interface Specifications**
- **PCI Express Base Specification 2.1 or 3.0**
- **Design Debugging Using the SignalTap II Embedded Logic Analyzer**

Setting Up Simulation

Changing the simulation parameters reduces simulation time and provides greater visibility.

Changing Between Serial and PIPE Simulation

By default, the Altera testbench runs a serial simulation. You can change between serial and PIPE simulation by editing the top-level testbench file.

The `hip_ctrl_simu_mode_pipe` signal and `enable_pipe32_sim_hwtcl` parameter, specify serial or PIPE simulation. When both are set to 1'b0, the simulation runs in serial mode. When both are set to 1'b1, the simulation runs in PIPE mode. Complete the following steps to enable PIPE simulation. These steps assume that the actual testbench in Gen1 x4 with an Avalon-MM 64-bit interface.

1. In the top-level testbench, which is **<*working_dir*>/<*variant*>/testbench/<*variant*>_*tb*/simulation/<*variant*>_tb.v**, change the signal, `hip_ctrl_simu_mode_pipe` to 1'b1 as shown:

   ```
   pcie_de_gen1_x4_ast64 pcie_de_gen1_x4_ast64_x_inst (.dut_hip_ctrl_simu_mode_pipe
   ( 1'b1 ),
   ```

2. In the top-level HDL module for the Hard IP which is **<*working_dir*>/<*variant*>/testbench/<*variant*>_tb/simulation/submodules/<*variant*>.v** change the parameter `enable_pipe32_sim_hwtcl` parameter to 1'b1 as shown:

```
altpcie_<dev>_hip_ast_hwtcl #( .enable_pipe32_sim_hwtcl ( 1 ),
```

Using the PIPE Interface for Gen1 and Gen2 Variants

Running the simulation in PIPE mode reduces simulation time and provides greater visibility.

Complete the following steps to simulate using the PIPE interface:

1. Change to your simulation directory, **<*work_dir*>/<*variant*>/testbench/<*variant*>_tb/simulation**
2. Open **<*variant*>_tb.v**.
3. Search for the string, `serial_sim_hwtcl`. Set the value of this parameter to 0 if it is 1.
4. Save **<*variant*>_tb.v**.

Reducing Counter Values for Serial Simulations

You can accelerate simulation by reducing the value of counters whose default values are set for hardware, not simulation.

Complete the following steps to reduce counter values for simulation:

1. Open **<*work_dir*>/<*variant*>/testbench/<*variant*>_tb/simulation/submodules/altpcie_tbed_<*dev*>_hwtcl.v** .
2. Search for the string, `test_in`.
3. To reduce the value of several counters, set `test_in[0] = 1`.
4. Save **altpcietb_bfm_top_rp.v**.

Disable the Scrambler for Gen1 and Gen2 Simulations

The encoding scheme implemented by the scrambler applies a binary polynomial to the data stream to ensure enough data transitions between 0 and 1 to prevent clock drift. The data is decoded at the other end of the link by running the inverse polynomial.

Complete the following steps to disable the scrambler:

1. Open **<*work_dir*>/<*variant*>/testbench/<*variant*>_tb/simulation/submodules/altpcie_tbed_<*dev*>_hwtcl.v**.
2. Search for the string, `test_in`.
3. To disable the scrambler, set `test_in[2] = 1`.
4. Save **altpcie_tbed_sv_hwtcl.v**.

Changing between the Hard and Soft Reset Controller

The Hard IP for PCI Express includes both hard and soft reset control logic. By default, Gen1 devices use the Hard Reset Controller. Gen2 devices use the soft reset controller. For variants that use the hard reset controller, changing to the soft reset controller provides greater visibility.

Complete the following steps to change to the soft reset controller:

1. Open **<*work_dir*>/<*variant*>/testbench/<*variant*>_tb/simulation/submodules/<*variant*>.v**.
2. Search for the string, `hip_hard_reset_hwtcl`.
3. If `hip_hard_reset_hwtcl = 1`, the hard reset controller is active. Set `hip_hard_reset_hwtcl = 0` to change to the soft reset controller.
4. Save **variant.v**.

Use Third-Party PCIe Analyzer

A third-party logic analyzer for PCI Express records the traffic on the physical link and decodes traffic, saving you the trouble of translating the symbols yourself. A third-party logic analyzer can show the two-way traffic at different levels for different requirements. For high-level diagnostics, the analyzer shows the LTSSM flows for devices on both side of the link side-by-side. This display can help you see the link training handshake behavior and identify where the traffic gets stuck. A traffic analyzer can display the contents of packets so that you can verify the contents. For complete details, refer to the third-party documentation.

BIOS Enumeration Issues

Both FPGA programming (configuration) and the initialization of a PCIe link require time. Potentially, an Altera FPGA including a Hard IP block for PCI Express may not be ready when the OS/BIOS begins enumeration of the device tree. If the FPGA is not fully programmed when the OS/BIOS begins its enumeration, the OS does not include the Hard IP for PCI Express in its device map.

You can use either of the following two methods to eliminate this issue:

- You can perform a soft reset of the system to retain the FPGA programming while forcing the OS/BIOS to repeat its enumeration.
- You can use CvP to program the device.

Transaction Layer Packet (TLP) Header Formats

A

2014.12.15

UG-01110_avst

Subscribe Send Feedback

The following figures show the header format for TLPs without a data payload.

For more information about the alignment of 3- and 4-dword headers refer to the related links below for *Data Alignment and Timing* for the Avalon-ST TX and RX Interfaces.

Figure A-1: Memory Read Request, 32-Bit Addressing

Memory Read Request, 32-Bit Addressing

	+0								+1								+2								+3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte 0	0	0	0	0	0	0	0	0	0	TC			0	0	0	0	TD	EP	Attr		0	0	Length									
Byte 4	Requester ID																Tag								Last BE				First BE			
Byte 8	Address[31:2]																														0	0
Byte 12	Reserved																															

Figure A-2: Memory Read Request, Locked 32-Bit Addressing

Memory Read Request, Locked 32-Bit Addressing

	+0								+1								+2								+3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte 0	0	0	0	0	0	0	0	1	0	TC			0	0	0	0	TD	EP	Attr		0	0	Length									
Byte 4	Requester ID																Tag								Last BE				First BE			
Byte 8	Address[31:2]																														0	0
Byte 12	Reserved																															

© 2014 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

Figure A-3: Memory Read Request, 64-Bit Addressing

Memory Read Request, 64-Bit Addressing

	+0								+1								+2								+3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte 0	0	0	0	0	0	0	0	0	0	TC			0	0	0	0	TD	EP	Attr		0	0	Length									
Byte 4	Requester ID																Tag								Last BE				First BE			
Byte 8	Address[63:32]																															
Byte 12	Address[31:2]																														0	0

Figure A-4: Memory Read Request, Locked 64-Bit Addressing

Memory Read Request, Locked 64-Bit Addressing

	+0								+1								+2								+3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte 0	0	0	1	0	0	0	0	1	0	TC			0	0	0	0	T	EP	Attr		0	0	Length									
Byte 4	Requester ID																Tag								Last BE				First BE			
Byte 8	Address[63:32]																															
Byte 12	Address[31:2]																														0	0

Figure A-5: Configuration Read Request Root Port (Type 1)

Configuration Read Request Root Port (Type 1)

	+0								+1								+2								+3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte 0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	TD	EP	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Byte 4	Requester ID																Tag								0	0	0	0	First BE			
Byte 8	Bus Number								Device No					Func			0	0	0	0	Ext Reg				Register No						0	0
Byte 12	Reserved																															

Figure A-6: I/O Read Request

I/O Read Request

	+0								+1								+2								+3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte 0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	TD	EP	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Byte 4	Requester ID																Tag								0	0	0	0	First BE			
Byte 8	Address[31:2]																														0	0
Byte 12	Reserved																															

Figure A-7: Message without Data

Message without Data

	+0								+1								+2								+3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte 0	0	0	1	1	0	r 2	r 1	r 0	0	TC			0	0	0	0	TD	EP	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Byte 4	Requester ID																Tag								Message Code							
Byte 8	Vendor defined or all zeros																															
Byte 12	Vendor defined or all zeros																															

Note

(1) Not supported in Avalon-MM.

Figure A-8: Completion without Data

Completion without Data

	+0								+1								+2								+3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte 0	0	0	0	0	1	0	1	0	0	TC			0	0	0	0	TD	EP	Attr		0	0	Length									
Byte 4	Completer ID																Status			B	Byte Count											
Byte 8	Requester ID																Tag								0	Lower Address						
Byte 12	Reserved																															

Figure A-9: Completion Locked without Data

Completion Locked without Data

	+0								+1								+2								+3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte 0	0	0	0	0	1	0	1	1	0	TC			0	0	0	0	TD	EP	Attr		0	0	Length									
Byte 4	Completer ID																Status			B	Byte Count											
Byte 8	Requester ID																Tag								0	Lower Address						
Byte 12	Reserved																															

Related Information

- **Data Alignment and Timing for the 64-Bit Avalon-ST RX Interface** on page 4-6
- **Data Alignment and Timing for the 128-Bit Avalon-ST RX Interface** on page 4-10
- **Data Alignment and Timing for 256-Bit Avalon-ST RX Interface**
- **Data Alignment and Timing for the 64-Bit Avalon-ST TX Interface** on page 4-18
- **Data Alignment and Timing for the 128-Bit Avalon-ST TX Interface** on page 4-20
- **Data Alignment and Timing for the 256-Bit Avalon-ST TX Interface**

TLP Packet Formats with Data Payload

Figure A-10: Memory Write Request, 32-Bit Addressing

Memory Write Request, 32-Bit Addressing

	+0								+1								+2								+3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte 0	0	1	0	0	0	0	0	0	0	TC			0	0	0	0	TD	EP	Attr		0	0	Length									
Byte 4	Requester ID																Tag								Last BE				First BE			
Byte 8	Address[31:2]																														0	0
Byte 12	Reserved																															

Figure A-11: Memory Write Request, 64-Bit Addressing

Memory Write Request, 64-Bit Addressing

	+0								+1								+2								+3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte 0	0	1	1	0	0	0	0	0	0	TC			0	0	0	0	TD	EP	Attr		0	0	Length									
Byte 4	Requester ID																Tag								Last BE				First BE			
Byte 8	Address[63:32]																															
Byte 12	Address[31:2]																														0	0

Figure A-12: Configuration Write Request Root Port (Type 1)

Configuration Write Request Root Port (Type 1)

	+0								+1								+2								+3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte 0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	TD	EP	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Byte 4	Requester ID																Tag								0	0	0	0	First BE			
Byte 8	Bus Number								Device No								0	0	0	0	Ext Reg				Register No						0	0
Byte 12	Reserved																															

Figure A-13: I/O Write Request

I/O Write Request

	+0								+1								+2								+3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte 0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	TD	EP	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Byte 4	Requester ID																Tag								0	0	0	0	First BE			
Byte 8	Address[31:2]																														0	0
Byte 12	Reserved																															

Figure A-14: Completion with Data

Completion with Data

	+0								+1								+2								+3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte 0	0	1	0	0	1	0	1	0	0	TC			0	0	0	0	TD	EP	Attr		0	0	Length									
Byte 4	Completer ID																Status				B	Byte Count										
Byte 8	Requester ID																Tag								0	Lower Address						
Byte 12	Reserved																															

Figure A-15: Completion Locked with Data

Completion Locked with Data

	+0								+1								+2								+3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte 0	0	1	0	0	1	0	1	1	0	TC			0	0	0	0	TD	EP	Attr		0	0	Length									
Byte 4	Completer ID																Status				B	Byte Count										
Byte 8	Requester ID																Tag								0	Lower Address						
Byte 12	Reserved																															

Figure A-16: Message with Data

Message with Data

	+0								+1								+2								+3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Byte 0	0	1	1	1	0	r2	r1	r0	0	TC			0	0	0	0	TD	EP	0	0	0	0	Length									
Byte 4	Requester ID																Tag								Message Code							
Byte 8	Vendor defined or all zeros for Slot Power Limit																															
Byte 12	Vendor defined or all zeros for Slots Power Limit																															

Lane Initialization and Reversal

B

2014.12.15

UG-01110_avst

Subscribe Send Feedback

Connected components that include IP blocks for PCI Express need not support the same number of lanes. The ×4 variations support initialization and operation with components that have 1, 2, or 4 lanes. The ×8 variant supports initialization and operation with components that have 1, 2, 4, or 8 lanes.

Lane reversal permits the logical reversal of lane numbers for the ×1, ×2, ×4, and ×8 configurations. Lane reversal allows more flexibility in board layout, reducing the number of signals that must cross over each other when routing the PCB.

Table B-1: Lane Assignments without Lane Reversal

Lane Number	7	6	5	4	3	2	1	0
×8 IP core	7	6	5	4	3	2	1	0
×4 IP core	—	—	—	—	3	2	1	0
×1 IP core	—	—	—	—	—	—	—	0

Table B-2: Lane Assignments with Lane Reversal

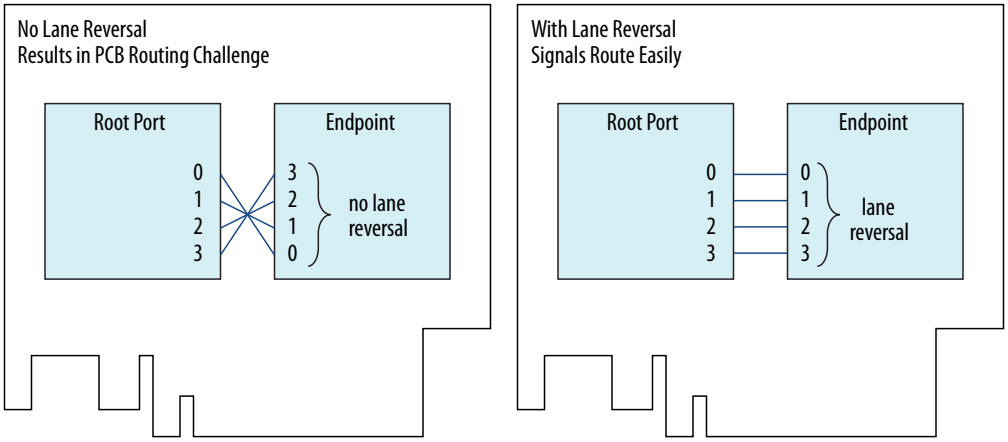
Core Config	8				4				1			
Slot Size	8	4	2	1	8	4	2	1	8	4	2	1
Lane pairings	7:0,6:1,5:2, 4:3, 3:4,2:5, 1:6,0:7	3:4,2:5, 1:6,0:7	1:6, 0:7	0:7	7:0,6:1, 5:2,4:3	3:0,2:1, 1:2,0:3	3:0, 2:1	3:0	7:0	3:0	1:0	0:0

© 2014 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

Figure B-1: Using Lane Reversal to Solve PCB Routing Problems

The following figure illustrates a PCI Express card with ×4 IP Root Port and a ×4 Endpoint on the top side of the PCB. Connecting the lanes without lane reversal creates routing problems. Using lane reversal solves the problem.



Additional Information

C

2014.12.15

UG-01110_avst | Subscribe | Send Feedback

Revision History for the Avalon-St Interface

Date	Version	Changes Made
2014.12.15	14.1	Made the following changes to the user guide: • In the figured titled *Specifying the Number of Transceiver Interfaces for Arria V and Cyclone V Devices*, removed the checkmark **Calibrate duty cycle during power up**. Duty cycle calibration occurs during Gen1 to Gen2 speed changes. • Corrected discussion of soft and hard reset controllers. The hardened reset controller is used for Arria V and Cyclone V devices. • Added simulation log file, **altpcie_monitor_<*dev*>_dlhip_tlp_file_log.log** in your simulation directory. Generation of the log file requires the following simulation file, **<*install_dir*>altera/altera_pcie/altera_pcie_a10_hip/altpcie_monitor_a10_dlhip_sim.sv**, that was not present in earlier releases of the Quartus II software. • Added statement that the bottom left hard IP block includes the CvP functionality for flip chip packages. For other package types, the CvP functionality is in the bottom right block. • Corrected bit definitions for `CvP Status` register. • Updated definition of `CVP_NUMCLKS` in the `CvP Mode Control` register. • Added definitions for `test_in[2]`, `test_in[6]` and `test_in[7]`. • For Cyclone V devices, changed recommendation to specify GT parts for Gen1 as well and Gen2 data rates.
2014.06.30	14.0	Made the following changes to the Cyclone V Hard IP for PCI Express: • Increased the size of `lmi_addr` to 15 bits.

© 2014 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, ENPIRION, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

ISO 9001:2008 Registered

Date	Version	Changes Made
		Made the following changes to the user guide: • Created separate user guides for variants using the Avalon-MM, Avalon-ST, and Avalon-MM with DMA interfaces to the Application Layer. • Added *Next Steps in Creating a Design for PCI Express* to *Datasheet* chapter. • Enhanced definition of Device ID and Sub-system Vendor ID to say that these registers are only valid in the Type 0 (Endpoint) Configuration Space. • Changed the default reset controller settings. By default Gen1 devices use the Hard Reset Controller. Gen2 uses the Soft Reset Controller. • Removed references to the MegaWizard® Plug-In Manager. In 14.0 the IP Parameter Editor Powered by Qsys has replaced the MegaWizard Plug-In Manager. • Removed reference to Gen2 x1 62.5 MHz configuration in *Application Layer Clock Frequency for All Combination of Link Width, Data Rate and Application Layer Interface Widths* table. This configuration is not supported. • Added definition for `test_in[6]test_out` bus. • Added definitions for Hard IP Reconfiguration bus. This bus became an optional feature in version 13.1 of the core. • Updated interrupt interface to reflect changes for multi-function support: `app_int_sts` implements legacy interrupts. • Added definitions for the `txmargin`, `txswing`, and `testin_zero` signals. • Removed definition for the `busy_xcvr_reconfig` signal which is not used. • Added section on relaxed ordering. • Added sections on making analog QSF and pin assignments. • Removed `reconfig_busy` port from connect between PHY IP Core for PCI Express and the Transceiver Reconfiguration Controller in the *Altera Transceiver Reconfiguration Controller Connectivity* figure. The Transceiver Reconfiguration Controller drives `reconfig_busy` port to the Altera PCIe Reconfig Driver. • Improved description of qword alignment of TLPs. • Added fact that DCD calibration is required for Gen2 data rate in the description of the transceiver reconfiguration signals. Updated figure showing Transceiver Reconfiguration Controller parameter editor.

Date	Version	Changes Made
		• Removed references to the ATX PLL. This PLL is not available for Cyclone V • Removed soft reset controller **.sdc** constraints from the ***<install_dir>*** **/ip/altera/altera_pcie/altera_pcie_hip_ast_ed/altpcied_<*dev*>.sdc** example. These constraints are now in a separate file in the **synthesis/submodules** directory. • Updated *Power Supply Voltage Requirements* table.
2014.12.20	13.1	Made the following changes: • Added constraints for `refclk` when CvP is enabled. • Corrected location information for `nPERSTL*`. • Corrected definition of `test_in[4:1]`. • In *Debugging* chapter, under changing between soft and hard reset controller, changed the file name in which the parameter `hip_hard_reset_hwtcl`must be set to 0 to use the soft reset controller. • Added explanation of channel labeling for serial data. The Hard IP on the left side of the device must connect to the appropriate channels on the left side of the device, and so on. • Added definition of `nreset_status` for variants using the Avalon-MM interface. • In *Transaction Layer Routing Rules and Programming Model for Avalon-MM Root Port* , added the fact that Type 0 Configuration Requests sent to the Root Port are not filtered by the device number. Application Layer software must filter out requests for device number greater than 0. • Added *Recommended Reset Sequence to Avoid Link Training Issues* to the *Debugging* chapter. • Added limitation for `RxmIrq_<n>_i[<m>:0]` when interrupts are received on consecutive cycles. • Updated timing diagram for `tl_cfg_ctl`. • Removed I/O Read Request and I/O Write Requests from TLPs supported for Avalon-MM interface. • Added note that the LTSSM interface can be used for SignalTap debugging. • Added restriction on the use of dynamic transceiver reconfiguration when CvP is enabled.
2014.05.06	13.0	Made the following changes:

Date	Version	Changes Made
		• Timing models are now final. • Added instructions for running the Single Dword variant. • Corrected definition of `test_in[4:1]`. This vector must be set to 4'b0100. • Corrected connection for `mgmt_clk_clk` in Figure 3-2. • Corrected definition of `nPERSTL*`. The device has 1 `nPERSTL*` pin for each instance of the Hard IP for PCI Express in the device. • Corrected feature comparison table in *Datasheet* chapter. The Avalon-MM Hard IP for PCI Express IP Core does not support legacy endpoints.

How to Contact Altera

To locate the most up-to-date information about Altera products, refer to the following table.

Contact (1)	Contact Method	Address
Technical support	Website	www.altera.com/support
Technical training	Website	www.altera.com/training
	Email	custrain@altera.com
Product literature	Website	www.altera.com/literature
Nontechnical support (general)	Email	nacomp@altera.com
(software licensing)	Email	authorization@altera.com

Note to Table:

1. You can also contact your local Altera sales office or sales representative.

Related Information

- **Technical Support**
- **Technical Training**
- **Customer Training**
- **Product Documentation**
- **Non-Technical Suport (general)**
- **Licensing**

Typographic Conventions

The following table shows the typographic conventions this document uses.

Table C-1: Visual CueMeaning

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Indicate command names, dialog box titles, dialog box options, and other GUI labels. For example, **Save As** dialog box. For GUI elements, capitalization matches the GUI.
bold type	Indicates directory names, project names, disk drive names, file names, file name extensions, software utility names, and GUI labels. For example, \ **qdesigns** directory, **D:** drive, and **chiptrip.gdf** file.
Italic Type with Initial Capital Letters	Indicate document titles. For example, *Stratix IV Design Guidelines*.
italic type	Indicates variables. For example, *n* + 1. Variable names are enclosed in angle brackets (< >). For example, <*file name*> and <*project name*> **.pof** file.
Initial Capital Letters	Indicate keyboard keys and menu names. For example, the Delete key and the Options menu.
"Subheading Title"	Quotation marks indicate references to sections in a document and titles of Quartus II Help topics. For example, "Typographic Conventions."
`Courier type`	Indicates signal, port, register, bit, block, and primitive names. For example, `data1`, `tdi`, and `input`. The suffix `n` denotes an active-low signal. For example, `resetn`. Indicates command line commands and anything that must be typed exactly as it appears. For example, `c:\qdesigns\tutorial\chiptrip.gdf`. Also indicates sections of an actual file, such as a Report File, references to parts of files (for example, the AHDL keyword `SUBDESIGN`), and logic function names (for example, `TRI`).

Visual Cue	Meaning
r	An angled arrow instructs you to press the Enter key.
1., 2., 3., anda., b., c., and so on	Numbered steps indicate a list of items when the sequence of the items is important, such as the steps listed in a procedure.
	Bullets indicate a list of items when the sequence of the items is not important.
1	The hand points to information that requires special attention.
h	The question mark directs you to a software help system with related information.
f	The feet direct you to another document or website with related information.
m	The multimedia icon directs you to a related multimedia presentation.
c	A caution calls attention to a condition or possible situation that can damage or destroy the product or your work.
w	A warning calls attention to a condition or possible situation that can cause you injury.
The **Subscribe** button links to the Email Subscription Management Center page of the Altera website, where you can sign up to receive update notifications for Altera documents.	
The **Feedback** icon allows you to submit feedback to Altera about the document. Methods for collecting feedback vary as appropriate for each document.	

Related Information

Email Subscription Management Center